**TRƯỜNG ĐẠI HỌC TÀI NGUYÊN VÀ MÔI TRƯỜNG TP.HCM**

**KHOA: HỆ THỐNG THÔNG TIN VÀ VIỄN THÁM**

**🙟🙞🕮🙜🙝**

A logo with a person and a book

Description automatically generated

**BÁO CÁO MÔN CÔNG NGHỆ LẬP TRÌNH ĐA NỀN TẢNG CHO ỨNG DỤNG DI ĐỘNG**

**NGÀNH CÔNG NGHỆ THÔNG TIN**

**ĐỀ TÀI: CHATAPP**

A logo with blue bubbles

Description automatically generated

Giảng viên hướng dẫn : **Th.S Nguyễn Thanh Truyền**

Sinh viên thực hiện: **Nguyễn Thị Thanh Nguyệt 0850080087**

**Phạm Lê Nhật Quang 0850080037**

**Nguyễn Thanh Tùng 0850080052**

**Hà Thị Thùy Trang 0850080050**

**Nguyễn Thành Đạt 0850080063**

**Trần Anh Tuấn 0850080111**

**Tô Tấn Tài 0850080102**

Lớp**: 08\_ĐH\_CNPM**

Khoá**: 2019-2023**

***TP. Hồ Chí Minh, tháng 08 năm 2023***

**TRƯỜNG ĐẠI HỌC TÀI NGUYÊN VÀ MÔI TRƯỜNG TP.HCM**

**KHOA: HỆ THỐNG THÔNG TIN VÀ VIỄN THÁM**

**🙟🙞🕮🙜🙝**

A logo with a person and a book

Description automatically generated

**BÁO CÁO MÔN CÔNG NGHỆ LẬP TRÌNH ĐA NỀN TẢNG CHO ỨNG DỤNG DI ĐỘNG**

**NGÀNH CÔNG NGHỆ THÔNG TIN**

**ĐỀ TÀI: CHATAPP**

A logo with blue bubbles

Description automatically generated

Giảng viên hướng dẫn : **Th.S Nguyễn Thanh Truyền**

Sinh viên thực hiện: **Nguyễn Thị Thanh Nguyệt 0850080087**

**Phạm Lê Nhật Quang 0850080037**

**Nguyễn Thanh Tùng 0850080052**

**Hà Thị Thùy Trang 0850080050**

**Nguyễn Thành Đạt 0850080063**

**Trần Anh Tuấn 0850080111**

**Tô Tấn Tài 0850080102**

Lớp**: 08\_ĐH\_CNPM**

Khoá**: 2019-2023**

***TP. Hồ Chí Minh, tháng 08 năm 2023***

**MỞ ĐẦU**

Ứng dụng chat là một trong những ứng dụng được sử dụng phổ biến nhất trên điện thoại di động. Để có thể xây dựng một ứng dụng chat hiệu quả, việc sử dụng kết nối real-time giữa client và server là rất quan trọng. Trong đề tài này, chúng ta sẽ tìm hiểu về lý thuyết việc xử lý kết nối, truyền tải dữ liệu real-time giữa client và server sử dụng giao thức websocket và thư viện SocketIO.

Bên cạnh đó, chúng ta cũng sẽ tìm hiểu về cách lấy dữ liệu từ server bằng cách sử dụng networking và HTTP, đồng thời cũng sẽ làm quen với định dạng dữ liệu JSON. Tất cả những kiến thức này sẽ được áp dụng để phát triển một ứng dụng chat sử dụng kết nối real-time giữa client và server, sử dụng framework Flutter và ngôn ngữ lập trình Dart.

Việc sử dụng framework Flutter cùng với ngôn ngữ lập trình Dart giúp cho việc phát triển một ứng dụng chat trở nên dễ dàng hơn bao giờ hết. Flutter cung cấp cho chúng ta một cách tiếp cận tiện lợi để xây dựng giao diện người dùng, đồng thời cũng cung cấp cho chúng ta các công cụ để xử lý dữ liệu real-time giữa client và server. Dart là một ngôn ngữ lập trình hiện đại, hỗ trợ nhiều tính năng tiên tiến và đơn giản để sử dụng.

Với những kiến thức được tìm hiểu trong đề tài này, chúng ta sẽ có thể phát triển một ứng dụng chat hiệu quả sử dụng kết nối real-time giữa client và server, đồng thời cũng sẽ nắm được những kiến thức quan trọng về networking, HTTP, JSON, websocket, Flutter và Dart.

**LỜI CẢM ƠN**

Chúng em xin gửi lời cảm ơn chân thành đến thầy về bài báo cáo môn Công nghệ lập trình đa nền tảng cho ứng dụng di động. Thầy đã cung cấp cho chúng em những kiến thức quý giá về lập trình ứng dụng di động trên nhiều nền tảng khác nhau, giúp chúng em hiểu rõ hơn về quy trình phát triển ứng dụng và các công nghệ mới nhất trong lĩnh vực này.

Thầy đã hướng dẫn cho chúng em cách sử dụng các công cụ như Flutter và ngôn ngữ dart để phát triển ứng dụng di động đa nền tảng, từ việc thiết kế giao diện đến lập trình logic và tương tác với dữ liệu. Thầy cũng giúp chúng em hiểu được cách tối ưu hóa hiệu suất và tăng tính ổn định của ứng dụng trên các nền tảng khác nhau.

Chúng em rất biết ơn sự tận tâm và nhiệt tình của thầy trong việc truyền đạt kiến thức cho tôi. Những kiến thức này sẽ rất hữu ích cho chúng em trong công việc phát triển ứng dụng di động trong tương lai.

Một lần nữa, chúng em xin chân thành cảm ơn thầy Nguyễn Thanh Truyền.

**NHẬN XÉT**

……………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………

………….., ngày….tháng….năm……

CÁN BỘ HƯỚNG DẪN

*(ký tên)*

**LỊCH LÀM VIỆC**

Thời gian làm bài từ ngày … tháng … năm 2023 đến ngày … tháng … năm 2023

|  |  |  |
| --- | --- | --- |
| **Thành viên** | **Nội dung công việc** | **Nhận xét** |
| **Nguyễn Thanh Tùng** | - Code chính(ngôn ngữ dart, framework fluter, IDE:Android studio, Server: NodeJS)  - Code chung với Quang:  + Code Server bằng NodeJS  + Code các chức để thực hiện việc call api lên server HTTP được viết bằng NodeJS để lấy dữ liệu là Json về cho các bạn khác xử lý thông tin lên màn hình giao diện. Config các kết nối tới database SQL. Code các chức năng để thực hiện kết nối dữ liệu chat RealTime giữa client và server thông qua webSocket server được tạo bằng Node.js và việc phân giải hình ảnh để có thể truyền tải lên server cũng như từ server gửi qua cho các client khác.  + Thống kê lại code và xữ lý lỗi khi gộp code. |  |
| **Phạm Lê Nhật Quang** | Code phụ (làm phụ code chính) Code chung với Tùng các chức năng như ở trên. |  |
| **Hà Thị Thùy Trang** | + Lý thuyết việc xử lý kết nối, dữ liệu RealTime giữa Client – Server giao thức websocket sử dụng thư viện SocketIO.  + Làm Word, PPT.  + Code màn hình giao diện các phòng chat và code xử lý chức năng (làm chung với Nguyệt phần code). |  |
| **Nguyễn Thành Đạt** | + Lý thuyết lấy dữ liệu từ server bằng cách call api sử dụng networking & HTTP, JSON.  + Code giao diện màn hình giao diện đăng nhập và code xử lý việc đăng nhập, chuyển màn hình.  + Code màn hình thông tin người dùng và code xử lý.  + Code màn hình thông tin người dùng và code xử lý.(Đạt, Tài, Tuấn code chung phần này). |  |
| **Tô Tấn Tài** | + Lý Thuyết lấy dữ liệu từ server bằng cách call api sử dụng networking & HTTP, JSON.  + Code màn hình giao diện đăng ký tài khoản cho các client và code xử lý việc đăng ký, chuyển màn hình.  + Code màn hình thông tin người dùng và code xử lý. .(Đạt, Tài, Tuấn code chung phần này). |  |
| **Trần Anh Tuấn** | + Lý Thuyết lấy dữ liệu từ server bằng cách call api sử dụng networking & HTTP, JSON.  + Code màn hình giao diện khung chat giữa các client thông qua server code xử lý nhận dữ liệu đổ lên giao diện.  + Code màn hình thông tin người dùng và code xử lý. .(Đạt, Tài, Tuấn code chung phần này). |  |
| **Nguyễn Thị Thanh Nguyệt** | + Lý Thuyết lấy dữ liệu từ server bằng cách call api sử dụng networking & HTTP, JSON.  + Xây dựng database cho app..  + code màn hình giao diện các phòng chat và code xử lý chức năng. |  |

TP. Hồ Chí Minh, ngày…tháng….năm 2023

**Sinh Viên**

**MỤC LỤC**

[**CHƯƠNG I** **TỔNG QUAN** 1](#_Toc142253419)

[**1.1** **Lý do chọn đề tài** 1](#_Toc142253420)

[**1.2** **Phạm vi đề tài** 1](#_Toc142253421)

[**1.3** **Mục tiêu và nhiệm vụ** 1](#_Toc142253422)

[**1.4** **Đối tượng phạm vi nghiên cứu** 2](#_Toc142253423)

[**1.5** **Hiện trạng của hệ thống** 2](#_Toc142253424)

[**CHƯƠNG II** **CƠ SỞ LÝ LUẬN** 4](#_Toc142253425)

[**2.1** **Lấy dữ liệu từ server bằng cách call api** 4](#_Toc142253426)

[**2.2** **Networking & HTTP, JSON** 5](#_Toc142253427)

[**2.2.1** **Networking & HTTP:** 5](#_Toc142253428)

[**2.2.2** **JSON (JavaScript Object Notation):** 5](#_Toc142253429)

[**2.3** **Giao thức websocket** 6](#_Toc142253430)

[**2.4** **Thư viện SocketIO** 7](#_Toc142253431)

[**2.5** **Truyền tải dữ liệu real-time giữa client và server.** 8](#_Toc142253432)

[**2.6** **Flutter** 9](#_Toc142253433)

[**2.6.1** **Flutter là gì?** 9](#_Toc142253434)

[**2.6.2** **Giới thiệu Flutter** 10](#_Toc142253435)

[**2.7** **Dart** 11](#_Toc142253436)

[**2.7.1** **Dart là gì** 11](#_Toc142253437)

[**2.7.2** **Tạo một Dart class đơn giản** 11](#_Toc142253438)

[**2.7.3** **Sử dụng các tham số tuỳ chọn** 13](#_Toc142253439)

[**2.7.4** **Tạo một factory** 15](#_Toc142253440)

[**CHƯƠNG III** **CÀI ĐẶT THỰC NGHIỆM** 19](#_Toc142253441)

[**3.1** **Cài đặt triển khai** 19](#_Toc142253442)

[**3.2** **Thiết kế cơ sở dữ liệu** 22](#_Toc142253443)

[**3.3** **Thiết kế các chức năng của app** 25](#_Toc142253444)

[**CHƯƠNG IV** **KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN** 38](#_Toc142253445)

[**4.1** **Kết luận** 38](#_Toc142253446)

[**4.2** **Hướng phát triển** 38](#_Toc142253447)

[**TÀI LIỆU THAM KHẢO** 40](#_Toc142253448)

**DANH MỤC CÁC BẢNG, SƠ ĐỒ, HÌNH**

[Hình 3. 1 Thêm dependencies 19](#_Toc142254073)

[Hình 3. 2 Ví dụ về cách kết nối đến server sử dụng SocketIO 20](#_Toc142254074)

[Hình 3. 3 ví dụ về cách kết nối đến server sử dụng WebSocket 20](#_Toc142254075)

[Hình 3. 4 Ví dụ về cách lấy dữ liệu từ server 21](#_Toc142254076)

[Hình 3. 5 Ví dụ về xử lý dữ liệu và thực hiện các tương tác 21](#_Toc142254077)

[Hình 3. 6 Đóng kết nối và giải phóng tài nguyên 22](#_Toc142254078)

[Hình 3. 7 ERD 22](#_Toc142254079)

[Hình 3. 8 Bảng message 23](#_Toc142254080)

[Hình 3. 9 Bảng conversation 23](#_Toc142254081)

[Hình 3. 10 Bảng Users 23](#_Toc142254082)

[Hình 3. 11 Cơ sở dữ liệu bản Message 24](#_Toc142254083)

[Hình 3. 12 Cơ sở dữ liệu bảng User 24](#_Toc142254084)

[Hình 3. 13 Cơ sở dữ liệu bảng conversation 24](#_Toc142254085)

[Hình 3. 14 call api http 25](#_Toc142254086)

[Hình 3. 15 Chat real time websocket 25](#_Toc142254087)

[Hình 3. 16 Địa chỉ IP 26](#_Toc142254088)

[Hình 3. 17 Sơ đồ tương tác của chức năng đăng ký 27](#_Toc142254089)

[Hình 3. 18 Sơ đồ hoạt động của chức năng đăng ký 27](#_Toc142254090)

[Hình 3. 19 Giao diện đăng ký 28](#_Toc142254091)

[Hình 3. 20 Mô hình hoạt động của chức năng đăng nhập 29](#_Toc142254092)

[Hình 3. 21 Mô hình tương tác của chức năng đăng nhập 30](#_Toc142254093)

[Hình 3. 22 Giao diện đăng nhập 30](#_Toc142254094)

[Hình 3. 23 Mô hình hoạt động của chức năng trò chuyện 31](#_Toc142254095)

[Hình 3. 24 Giao diện cuộc trò chuyện 32](#_Toc142254096)

[Hình 3. 25 Giao diện danh sách người dùng 34](#_Toc142254097)

[Hình 3. 26 Giao diện tạo phòng chat nhóm 34](#_Toc142254098)

[Hình 3. 27 Giao diện trang cá nhân 36](#_Toc142254099)

[Hình 3. 28 Giao diện đổi tên người dùng 38](#_Toc142254100)

**KÍ HIỆU CÁC CỤM TỪ VIẾT TẮT**

1. **TỔNG QUAN**

## **Lý do chọn đề tài**

Thứ nhất, ứng dụng chat là một trong những ứng dụng phổ biến nhất trên điện thoại di động và web hiện nay. Việc tìm hiểu về cách xử lý kết nối và truyền tải dữ liệu real-time giữa client và server là rất quan trọng để đảm bảo tính nhanh chóng và tin cậy trong truyền tải dữ liệu, đồng thời cũng giúp cho người dùng có thể nhận được thông báo ngay lập tức khi có tin nhắn mới.

Thứ hai, sử dụng giao thức websocket và thư viện SocketIO giúp cho việc truyền tải dữ liệu real-time giữa client và server trở nên dễ dàng và hiệu quả hơn. SocketIO hỗ trợ nhiều tính năng tiên tiến và dễ sử dụng, giúp cho việc xây dựng ứng dụng chat trở nên dễ dàng hơn.

Thứ ba, việc sử dụng networking, HTTP và JSON để lấy dữ liệu từ server là rất quan trọng trong phát triển ứng dụng. Chúng ta có thể sử dụng các API được cung cấp bởi server để gửi các yêu cầu tới server và nhận các phản hồi từ server, sau đó sử dụng các thư viện để chuyển đổi dữ liệu từ dạng JSON sang các đối tượng trong ứng dụng của chúng ta.

Thứ tư, việc sử dụng Flutter và ngôn ngữ Dart giúp cho việc phát triển ứng dụng chat trở nên dễ dàng và tiện lợi hơn. Flutter cung cấp cho chúng ta một cách tiếp cận tiện lợi để xây dựng giao diện người dùng, đồng thời cũng cung cấp cho chúng ta các công cụ để xử lý dữ liệu real-time giữa client và server. Dart là một ngôn ngữ lập trình hiện đại, hỗ trợ nhiều tính năng tiên tiến và đơn giản để sử dụng.

## **Phạm vi đề tài**

Tìm hiểu về giao thức websocket và thư viện SocketIO để truyền tải dữ liệu real-time giữa client và server.

Tìm hiểu về networking, HTTP và JSON để lấy dữ liệu từ server bằng cách call api.

Thiết kế và xây dựng ứng dụng chatapp bằng Flutter và Dart.

Kiểm thử và đánh giá tính năng, hiệu năng của ứng dụng.

## **Mục tiêu và nhiệm vụ**

Nhiệm vụ của đề tài bao gồm:

Tìm hiểu về giao thức websocket và thư viện SocketIO để truyền tải dữ liệu real-time giữa client và server.

Tìm hiểu về networking, HTTP và JSON để lấy dữ liệu từ server bằng cách call api.

Thiết kế giao diện cho ứng dụng chatapp.

Xây dựng chức năng đăng nhập và đăng ký tài khoản.

Xây dựng chức năng chat giữa hai người dùng.

Xây dựng chức năng chat nhóm.

Xây dựng chức năng gửi tệp tin và ảnh trong cuộc trò chuyện.

Kiểm thử và đánh giá tính năng, hiệu năng của ứng dụng.

## **Đối tượng phạm vi nghiên cứu**

Đối tượng nghiên cứu

Đối tượng của đề tài là các nhà phát triển ứng dụng di động và web có nhu cầu tìm hiểu về cách xử lý kết nối và truyền tải dữ liệu real-time giữa client và server sử dụng giao thức websocket và thư viện SocketIO, cùng với việc lấy dữ liệu từ server bằng cách call api sử dụng networking & HTTP, JSON để phát triển ứng dụng chatapp bằng Flutter và Dart.

Phạm vi nghiên cứu

tìm hiểu, thiết kế và xây dựng ứng dụng chatapp sử dụng giao thức websocket và thư viện SocketIO để truyền tải dữ liệu real-time giữa client và server, cùng với việc lấy dữ liệu từ server bằng cách call api sử dụng networking & HTTP, JSON. Ứng dụng sẽ được xây dựng bằng Flutter và Dart và sẽ bao gồm các chức năng như đăng nhập, đăng ký tài khoản, chat giữa hai người dùng, chat nhóm, gửi tệp tin và ảnh trong cuộc trò chuyện.

Đề tài sẽ không đi sâu vào các chi tiết kỹ thuật của các công nghệ sử dụng trong đề tài, mà tập trung vào cách sử dụng chúng để thiết kế và xây dựng ứng dụng chatapp hiệu quả.

## **Hiện trạng của hệ thống**

Hiện nay, có nhiều hệ thống chatapp được sử dụng rộng rãi trên các nền tảng di động và web. Một số hệ thống phổ biến nhất bao gồm:

1. WhatsApp: Là một ứng dụng nhắn tin và gọi điện miễn phí được sở hữu bởi Facebook. WhatsApp sử dụng giao thức Signal để mã hóa tin nhắn và tài liệu của người dùng.

2. Facebook Messenger: Là ứng dụng nhắn tin và gọi điện được sở hữu bởi Facebook. Facebook Messenger cung cấp tính năng chat, gọi video, gửi tệp tin và hỗ trợ chatbot.

3. WeChat: Là một ứng dụng nhắn tin và gọi điện phổ biến tại Trung Quốc. WeChat cung cấp tính năng tương tự như Facebook Messenger và hỗ trợ thanh toán trực tuyến.

4. Slack: Là một ứng dụng chat dành cho doanh nghiệp và nhóm làm việc. Slack cung cấp tính năng chat, gọi video và tích hợp với các công cụ làm việc khác như Google Drive và Trello.

5. Discord: Là một ứng dụng chat dành cho cộng đồng game thủ. Discord cung cấp tính năng chat, gọi video và cho phép người dùng tạo máy chủ riêng để tương tác với cộng đồng của mình.

Tất cả các hệ thống chatapp trên đều sử dụng các công nghệ để truyền tải dữ liệu real-time giữa client và server, đồng thời hỗ trợ các tính năng như chat, gọi video, gửi tệp tin và tích hợp với các công cụ làm việc khác. Tuy nhiên, mỗi hệ thống có các đặc điểm riêng và phục vụ cho mục đích sử dụng khác nhau.

1. **CƠ SỞ LÝ LUẬN**
2. **Lấy dữ liệu từ server bằng cách call api**

Trong Flutter, "call API" có ý nghĩa là gửi yêu cầu (request) đến một địa chỉ (endpoint) cụ thể trên mạng để lấy dữ liệu hoặc thực hiện một hành động nào đó từ một dịch vụ hoặc ứng dụng khác.

Để "call API" trong Flutter, bạn có thể sử dụng các gói thư viện hỗ trợ như `http` hoặc `dio`. Cả hai gói này đều cung cấp phương thức để gửi các yêu cầu HTTP như GET, POST, PUT, DELETE đến server và xử lý phản hồi từ server.

Ví dụ, để gửi một yêu cầu GET đến một API endpoint và lấy dữ liệu từ server, bạn có thể làm như sau sử dụng gói `http`:

1. Thêm gói thư viện vào file `pubspec.yaml`:

dependencies:

http: ^0.13.4

2. Import các thư viện cần thiết:

import 'package:http/http.dart' as http;

3. Gửi yêu cầu GET và xử lý phản hồi:

A computer screen with white and green text

Description automatically generated

Trong ví dụ trên, chúng ta sử dụng phương thức `http.get()` để gửi yêu cầu GET đến endpoint API đã xác định. Sau đó, chúng ta kiểm tra mã trạng thái của phản hồi từ server. Nếu mã trạng thái là 200 (mã thành công), chúng ta lấy dữ liệu từ `response.body` và tiếp tục xử lý và sử dụng dữ liệu theo nhu cầu. Trong trường hợp mã trạng thái không phải là 200, chúng ta in ra thông báo lỗi.

Lưu ý rằng ví dụ trên chỉ đơn giản để minh họa cách gọi API trong Flutter. Bạn cần kiểm tra tài liệu của API cụ thể mà bạn đang sử dụng để biết các chi tiết về endpoint, phương thức HTTP, tham số truy vấn, xác thực, và cách xử lý phản hồi từ server.

1. **Networking & HTTP, JSON**
2. **Networking & HTTP:**

* Networking: Networking trong Flutter là quá trình gửi và nhận dữ liệu giữa ứng dụng Flutter và server thông qua mạng Internet. Điều này cho phép ứng dụng của bạn tương tác với các dịch vụ web, API và server khác để lấy dữ liệu, gửi dữ liệu, cập nhật dữ liệu và xóa dữ liệu.
* HTTP: HTTP (Hypertext Transfer Protocol) là một giao thức mạng dựa trên yêu cầu/đáp ứng (request/response) được sử dụng rộng rãi trên Internet. Trong Flutter, bạn có thể sử dụng các gói thư viện như `http` hoặc `dio` để gửi yêu cầu HTTP đến server và nhận phản hồi từ server. Các phương thức HTTP phổ biến như GET, POST, PUT và DELETE được sử dụng để thực hiện các hoạt động khác nhau như truy vấn dữ liệu, gửi dữ liệu và cập nhật dữ liệu.

1. **JSON (JavaScript Object Notation):**

* JSON: JSON là một định dạng dữ liệu phổ biến được sử dụng để truyền tải dữ liệu giữa client và server. Nó dựa trên cú pháp JavaScript và được xem là dễ đọc và dễ hiểu cho con người. JSON có thể chứa các cấu trúc dữ liệu như mảng (array) và đối tượng (object), giúp đại diện cho dữ liệu phức tạp.
* Dart JSON: Trong Flutter, bạn có thể sử dụng gói `dart:convert` để chuyển đổi giữa các đối tượng Dart và chuỗi JSON. Gói này cung cấp các phương thức như `json.encode()` để chuyển đổi đối tượng Dart thành chuỗi JSON và `json.decode()` để chuyển đổi chuỗi JSON thành đối tượng Dart. Điều này cho phép bạn gửi và nhận dữ liệu theo định dạng JSON thông qua yêu cầu HTTP và xử lý dữ liệu JSON trong ứng dụng của mình.

Kết hợp Networking & HTTP và JSON, bạn có thể gửi yêu cầu HTTP đến server thông qua Networking, nhận phản hồi từ server dưới dạng chuỗi JSON, sau đó chuyển đổi chuỗi JSON thành các đối tượng Dart để xử lý và hiển thị dữ liệu trong ứng dụng Flutter của mình. Điều này cho phép ứng dụng của bạn tương tác với các dịch vụ web và server khác một cách linh hoạt và truyền tải dữ liệu theo định dạng JSON phổ biến.

1. **Giao thức websocket**

Giao thức WebSocket trong Flutter là một giao thức truyền tải dữ liệu hai chiều và toàn cầu, cho phép thiết lập kết nối liên tục và tương tác thông qua một kết nối duy nhất giữa client và server. Giao thức này được thiết kế để đảm bảo việc truyền tải dữ liệu hiệu quả và thời gian thực, thay vì yêu cầu client phải gửi yêu cầu và server trả về phản hồi như trong giao thức HTTP truyền thống.

WebSocket sử dụng mô hình sự kiện và thông báo, cho phép server và client gửi và nhận dữ liệu một cách không đồng bộ. Khi kết nối WebSocket được thiết lập, cả client và server có thể gửi dữ liệu cho nhau mà không cần chờ phản hồi từ phía đối tác. Điều này làm giảm đáng kể độ trễ và tăng tính thời gian thực của ứng dụng.

Giao thức WebSocket được định nghĩa trong các tiêu chuẩn của IETF (Internet Engineering Task Force), cụ thể là RFC 6455. Nó được thiết kế như một phần của HTML5 và hỗ trợ bởi hầu hết các trình duyệt hiện đại. WebSocket cũng được hỗ trợ trong nhiều ngôn ngữ lập trình và framework, bao gồm cả Flutter.

Dưới đây là một số ví dụ về ứng dụng Flutter sử dụng giao thức WebSocket:

* Ứng dụng chat real-time: Đây là một ví dụ phổ biến về việc sử dụng WebSocket trong ứng dụng Flutter. Ứng dụng chat real-time cho phép người dùng gửi và nhận tin nhắn trong thời gian thực, mà không cần làm mới trang hoặc gửi yêu cầu HTTP. WebSocket giúp duy trì kết nối liên tục giữa client và server, cho phép các tin nhắn mới được gửi và hiển thị ngay lập tức trên giao diện người dùng.
* Ứng dụng theo dõi giá trị chứng khoán: Một ứng dụng Flutter có thể sử dụng WebSocket để theo dõi và hiển thị thông tin giá trị của các chứng khoán. Thông tin sẽ được cập nhật theo thời gian thực từ server thông qua kết nối WebSocket, cho phép người dùng xem các thay đổi giá trị mới nhất mà không cần làm mới trang hoặc gửi yêu cầu HTTP.
* Ứng dụng đồng bộ hóa dữ liệu trực tuyến: Đôi khi, trong một ứng dụng Flutter, bạn có thể muốn đồng bộ hóa dữ liệu giữa nhiều người dùng hoặc thiết bị khác nhau. Sử dụng WebSocket, bạn có thể thiết lập kết nối liên tục giữa client và server để truyền tải và cập nhật dữ liệu trực tuyến. Khi có sự thay đổi dữ liệu từ một người dùng hoặc thiết bị, dữ liệu sẽ được gửi đến các người dùng/thiết bị khác qua kết nối WebSocket để đồng bộ hóa.
* Ứng dụng trò chơi real-time đa người chơi: Sử dụng giao thức WebSocket, bạn có thể xây dựng ứng dụng trò chơi real-time đa người chơi trong Flutter. WebSocket cho phép các người chơi gửi và nhận các tương tác trò chơi như di chuyển, tương tác với đối tượng và trò chuyện trong thời gian thực. Điều này tạo ra trải nghiệm trò chơi liên tục và mượt mà, mà không cần phải làm mới trang hoặc gửi yêu cầu HTTP.

Đây chỉ là một số ví dụ cơ bản về cách sử dụng giao thức WebSocket trong ứng dụng Flutter. Tuy nhiên, có nhiều ứng dụng khác mà bạn có thể xây dựng sử dụng WebSocket tùy thuộc vào yêu cầu cụ thể của dự án của bạn.

1. **Thư viện SocketIO**

Thư viện SocketIO trong Flutter là một thư viện giúp kết nối và giao tiếp với máy chủ sử dụng Socket.IO, một thư viện giao tiếp real-time hai chiều giữa client và server. Socket.IO sử dụng giao thức WebSocket và có khả năng tự động chuyển đổi sang các giao thức truyền thống khác nếu WebSocket không được hỗ trợ trong môi trường hiện tại.

Thư viện SocketIO trong Flutter cung cấp các lớp và phương thức cho phép thiết lập kết nối và gửi/nhận sự kiện real-time giữa client và server. Điều này cho phép ứng dụng Flutter giao tiếp thông qua kết nối real-time với máy chủ Socket.IO, nhận thông báo và dữ liệu mới ngay khi chúng được gửi từ phía server.

Thư viện SocketIO trong Flutter được lấy từ nguồn chính thức là `flutter\_socket\_io`. Để sử dụng thư viện này, bạn có thể thêm gói `flutter\_socket\_io` vào file `pubspec.yaml`:

dependencies:

flutter\_socket\_io: ^0.7.1

Sau khi thêm gói thành công, bạn có thể import thư viện:

import 'package:flutter\_socket\_io/flutter\_socket\_io.dart';

import 'package:flutter\_socket\_io/socket\_io\_manager.dart';

Thư viện `flutter\_socket\_io` cung cấp các lớp như `SocketIO`, `SocketIOManager` và các phương thức để thiết lập kết nối, gửi/nhận sự kiện và quản lý kết nối Socket.IO trong ứng dụng Flutter.

Ví dụ về việc sử dụng thư viện SocketIO trong Flutter:

SocketIO socketIO = SocketIOManager().createSocketIO('https://example.com', '/');

socketIO.init();

socketIO.connect();

socketIO.onConnect((data) {

print('Connected');

});

socketIO.on('message', (data) {

print('Received message: $data');

});

socketIO.sendMessage('Hello Server!', 'message');

socketIO.disconnect();

Trong ví dụ trên, chúng ta tạo một đối tượng `SocketIO` và thiết lập kết nối đến máy chủ Socket.IO tại `'https://example.com'`. Sau khi thiết lập kết nối, chúng ta đăng ký các bộ lắng nghe sự kiện (`onConnect` và `on`) để nhận và xử lý dữ liệu từ server. Cuối cùng, chúng ta gửi một tin nhắn đến server bằng cách sử dụng phương thức `sendMessage` và ngắt kết nối bằng cách gọi `disconnect`.

Lưu ý rằng, để sử dụng thư viện SocketIO trong Flutter, bạn cần có một máy chủ hỗ trợ Socket.IO để giao tiếp với ứng dụng của bạn.

1. **Truyền tải dữ liệu real-time giữa client và server.**

Truyền tải dữ liệu real-time giữa client và server trong Flutter được thực hiện thông qua việc sử dụng giao thức WebSocket và thư viện SocketIO.

WebSocket là một giao thức truyền tải dữ liệu hai chiều thông qua một kết nối TCP duy nhất. Nó cho phép thiết lập kết nối liên tục giữa client và server và truyền tải dữ liệu real-time mà không cần phải tạo các yêu cầu HTTP mới. Giao thức WebSocket được hỗ trợ trên hầu hết các trình duyệt và máy chủ hiện đại.

Thư viện SocketIO là một thư viện giao tiếp real-time hai chiều giữa client và server, sử dụng giao thức WebSocket và có khả năng tự động chuyển đổi sang các giao thức truyền thống khác nếu WebSocket không được hỗ trợ trong môi trường hiện tại. Thư viện SocketIO trong Flutter cung cấp các lớp và phương thức để thiết lập kết nối, gửi/nhận sự kiện và quản lý kết nối Socket.IO trong ứng dụng.

Để sử dụng truyền tải dữ liệu real-time giữa client và server trong Flutter, bạn có thể sử dụng thư viện SocketIO. Thư viện SocketIO trong Flutter có thể được lấy từ nguồn chính thức là `flutter\_socket\_io`. Bạn có thể thêm gói `flutter\_socket\_io` vào file `pubspec.yaml` để sử dụng thư viện:

dependencies:

flutter\_socket\_io: ^0.7.1

Sau khi thêm gói thành công, bạn có thể import thư viện và sử dụng các lớp và phương thức của nó để thiết lập kết nối và truyền tải dữ liệu real-time giữa client và server.

Ví dụ, bạn có thể sử dụng phương thức `sendMessage` để gửi dữ liệu từ client đến server:

socketIO.sendMessage('Hello Server!', 'message');

Và bạn có thể sử dụng sự kiện `on` để lắng nghe dữ liệu từ server:

socketIO.on('message', (data) {

print('Received message: $data');

});

Thông qua các phương thức và sự kiện như trên, bạn có thể truyền tải dữ liệu real-time giữa client và server trong Flutter sử dụng thư viện SocketIO.

1. **Flutter**

### **Flutter là gì?**

**Flutter** là một **framework** mã nguồn mở cho phép tạo ứng dụng di động với hiệu năng cao, chất lượng tốt hỗ trợ đa nền tảng, phù hợp với phát triển ứng dụng Android và iOS.

 Sử dụng ngôn ngữ Dart của chính Google, Flutter rất dễ học, mạnh mẽ, hiệu năng cao và phát triển ứng dụng di động một cách nhanh chóng.

Trong khoá học này, mình sẽ giúp các bạn làm quen với Flutter framework, hướng dẫn cài đặt Flutter SDK, thiết lập Android Studio để xây dựng một ứng dụng Flutter căn bản, nắm vững kỹ thuật của  Flutter framework và có khả năng phát triển các loại ứng dụng khác nhau sử dụng Flutter framework.

### **Giới thiệu Flutter**

Nhìn chung phát triển ứng dụng di động là một công việc phức tạp và nhiều khó khăn. Có rất nhiều framework hỗ trợ bạn phát triển một ứng dụng mobile. Android cung cấp một framework cơ bản dựa trên ngôn ngữ lập trình Java còn iOS thì cung cấp framework dựa trên Objective-C / Swift

Tuy nhiên hầu hết các ứng dụng hiện nay, đều hỗ trợ cả 2 nền tảng Android và iOS, do đó cùng lúc phát triển 2 dự án khác nhau với 2 framework khác nhau là một công việc phức tạp và lãng phí thời gian công sức. Do đó người ta đã phát triển các framework lập trình đa nền tảng để giải quyết vấn đề này. Một framework rất phổ biến hiện nay là React Native được phát triển bới Facebook đang được sử dụng rất rộng rãi. Tuy nhiên React Native vẫn thông qua các api của các framework gốc như Android hay iOS do đó bị hạn chế và tốc độ kém.

Như một sự phát triển của tương lai, Flutter được phát triển bới chính Google, đơn vị sở hữu Android như một đối trọng trực tiếp với React Native. Thay vì gọi các api của framework gốc, Flutter tạo ra giao diện trực tiếp từ api của hệ điều hành. Nhờ đó ứng dụng sẽ chạy nhanh hơn, mượt mà hơn và đẹp hơn.

Flutter cung cấp rất nhiều widgets (UI) là các thành phần đồ hoạ được thiết kế riêng. Những đối tượng đồ hoạ này được tối ưu phù hợp với môi trường mobile và dễ dàng trong việc thiết kế như HTML.

Cụ thể, ứng dụng Flutter sẽ sử dụng các widget riêng. Flutter widgets cung cấp các animations (hiệu ứng) và gestures (thao tác) riêng. Ứng dụng được phát triển dựa trên logic của  reactive programming. Mỗi Widget sẽ có rất nhiều trang thái. Bằng cách thay đổi trạng thái của widget, Flutter sẽ tự động (reactive programming) so sánh trạng thái của widget (cũ và mới) để tạo ra những thay đổi cần thiết thay vì khởi tạo lại cả đối tượng.

Mình sẽ nói kỹ hơn về kĩ thuật này trong các bài tiếp theo

#### **Tính năng của Flutter**

Flutter framework có những đặc điểm sau

* Hiện đã và là một react framework
* Sử dụng ngôn ngữ lập trình Dart đơn giản và dễ học
* Phát triển ứng dụng nhanh
* Giao diện người dùng rất đẹp và linh hoạt
* Hỗ trợ rất nhiều widget khác nhau
* Thể hiện cùng một UI trên nhiều nền tảng
* Ứng dụng có hiệu năng cao

#### **Điểm mạnh của Flutter**

Flutter đi kèm với nhiều widget đẹp và có độ tuỳ biến cao giúp phát triển ứng dụng hiệu năng cao vượt trội đáp ứng mọi nhu cầu và tuỳ biến. Bên cạnh đó Flutter còn có những điểm mạnh sau:

* *Dart* có một kho lớn các gói phần mềm cho phép bạn mở rộng khả năng cho ứng dụng của mình
* Các lập trình viên chỉ cần viết một chương trình duy nhất cho tất cả các ứng dụng (Android và iOS) . *Flutter* có thể mở rộng ra các nền tảng khác trong thời gian tới.
* Flutter dễ dàng kiểm thử hơn do tiết kiệm thời gian kiểm thử trên từng nền tảng.
* Nhờ sự đơn giản của mình, Flutter là lựa chọn hàng đầu cho các ứng dụng mới. Nó còn dễ dàng tuỳ biến và mở rộng lên càng mạnh mẽ hơn
* Với Flutter, lập trình viên có toàn quyền để sắp xếp bổ trí điều khiển các widget
* Flutter có bộ công cụ phát triển (developer tools) rất hoàn thiện và đầy đủ, đặc biệt với tính năng hot reload đẩy nhanh tốc độ build ứng dụng đáng kinh ngạc

1. **Dart**

### **Dart là gì**

Dart là ngôn ngữ lập trình cho Flutter- bộ công cụ giao diện người dùng của Google để xây dựng các ứng dụng Mobile, Web và Desktop app đẹp, được biên dịch nguyên bản từ một cơ sở mã code duy nhất.

Theo cá nhân mình thì Dart giống Java 70 ~ 80%, vì vậy nếu những bạn nào đã sử dụng Java thì quá tiện. Với những bạn thấy nó mới tinh, hay là ngôn ngữ lập trình đầu tiên của mình thì, không thành vấn đề bắt đầu cũng ok thôi

### **Tạo một Dart class đơn giản**

Class Bicycle chứa một vài private instance variable với getters và setters. Hàm main() khởi tạo Bicycle và in nó ra trong console.

Mọi người có thể viết đoạn code dưới đây và cùng phân tích nó

class Bicycle {

int cadence;

int \_speed = 0;

int get speed => \_speed;

int gear;

Bicycle(this.cadence, this.gear);

// phanh

void applyBrake(int decrement) {

\_speed -= decrement;

}

// tăng tốc

void speedUp(int increment) {

\_speed += increment;

}

@override

String toString() => 'Bicycle: $\_speed mph';

}

void main() {

var bike = Bicycle(2, 1);

print(bike);

}

Phương thức chính của Dart được đặt tên là main (). Nếu bạn cần quyền truy cập vào các đối số dòng lệnh, bạn có thể thêm

main(List<String> args)

Phương thức main () sống ở cấp cao nhất. Trong Dart, bạn có thể xác định mã bên ngoài các lớp. Tất cả các biến, hàm, getters và setters đều có thể tồn tại bên ngoài các lớp.

Cả main () và Bicycle đều không được khai báo là public vì tất cả các định danh đều là public theo mặc định. Dart không có các từ khóa cho public, private hoặc protected.

Nếu Class Bicycle có 3 tham số : cadence, speed và gear, ta có thể tạo constructor như sau

Bicycle(this.cadence, this.speed, this.gear);

constructors này không có phần thân và có dấu ; cuối, sử dụng cách này khá là tiện dụng để gán giá trị cho instance variables. Đoạn mã trên tương đương với đoạn code sau (như Java)

Bicycle(int cadence, int speed, int gear) {

this.cadence = cadence;

this.speed = speed;

this.gear = gear;

}

Do biến speed ta muốn nó là private và chỉ có quyền đọc thì ta thêm dấu \_ đằng trước, khởi tạo nó bằng 0 và thêm một phương thức get với dấu => như trên. Nên trong constructor ta remove biến này đi. Dart compiler nhận biết những biến hay thư viện nào được khai báo là dấu gạch dưới là private.

Không giống như Java mà như Kotlin, Dart cung cấp getters và setters ngầm định cho tất cả các biến công khai (public). Ta không cần phải xác định getters hoặc setters của riêng mình trừ khi ta muốn thực thi các biến chỉ đọc hoặc ghi, tính toán hoặc xác minh một giá trị ở nơi khác.

### **Sử dụng các tham số tuỳ chọn**

Trong Java nếu bạn tạo nhiều constructors cho một class thì các constructors này sẽ cũng tên với class nhưng khác nhau về số lượng hoặc kiểu tham số, sẽ như thế này:

public class Rectangle {

public int width = 0;

public int height = 0;

public Point origin;

// four constructors

public Rectangle() {

origin = new Point(0, 0);

}

public Rectangle(Point p) {

origin = p;

}

public Rectangle(int w, int h) {

origin = new Point(0, 0);

width = w;

height = h;

}

public Rectangle(Point p, int w, int h) {

origin = p;

width = w;

height = h;

}

// a method for moving the rectangle

public void move(int x, int y) {

origin.x = x;

origin.y = y;

}

// a method for computing the area of the rectangle

public int getArea() {

return width \* height;

}

}

Dart không làm như thế này mà sẽ là:

Rectangle({this.origin = const Point(0, 0), this.width = 0, this.height = 0});

chỉ với một dòng nó đã khai báo được 1 class và thay thế 4 constructors trong Java. this.origin, this.width và this.height là các tham số tùy chọn, có khi khi tạo instance thì có hay không có cũng được. Chúng được đặt trong dấu ngoặc nhọn. Cú pháp this.origin = const Point (0, 0) chỉ định giá trị mặc định của Point (0,0) cho instance variable gốc.

Chúng ta có thể override hàm toString()

@override

String toString() =>

'Origin: (${origin.x}, ${origin.y}), width: $width, height: $height';

Viết code trong hàm main:

main() {

print(Rectangle(origin: const Point(10, 20), width: 100, height: 200));

print(Rectangle(origin: const Point(10, 10)));

print(Rectangle(width: 200));

print(Rectangle());

}

Và chạy để xem output:

Origin: (10, 20), width: 100, height: 200

Origin: (10, 10), width: 0, height: 0

Origin: (0, 0), width: 200, height: 0

Origin: (0, 0), width: 0, height: 0

### **Tạo một factory**

Factories, một mẫu thiết kế thường được sử dụng trong Java, có một số lợi thế so với việc khởi tạo đối tượng trực tiếp, chẳng hạn như ẩn các chi tiết của việc khởi tạo, cung cấp khả năng trả về một kiểu con của kiểu trả về của factory và tùy chọn trả về một đối tượng hiện có thay vì một đối tượng mới.

Ở đây tôi sẽ trình bày 2 cách để tạo một shape-creation factory:

Tạo một top-level function.

Tạo một factory constructor.

Ta sử dụng ví dụ Shapes, ví dụ này khởi tạo các shapes và in tính toán diện tích của chúng:

import 'dart:math';

abstract class Shape {

num get area;

}

class Circle implements Shape {

final num radius;

Circle(this.radius);

num get area => pi \* pow(radius, 2);

}

class Square implements Shape {

final num side;

Square(this.side);

num get area => pow(side, 2);

}

main() {

final circle = Circle(2);

final square = Square(2);

print(circle.area);

print(square.area);

}

Đây là output

12.566370614359172

4

Dart có hỗ trợ abstract class

Có thể define nhiều class trong một file

Ngoài dart:match thì dart:core, dart:async, dart:convert, và dart:collection cũng là các thư viện cốt lõi của Dart

Option 1: Tạo một top-level function, tức function không thuộc một Class nào

Shape shapeFactory(String type) {

if (type == 'circle') return Circle(2);

if (type == 'square') return Square(2);

throw 'Can\'t create $type.';

}

gọi hàm factory bằng cách thay thế hai dòng đầu tiên trong phương thức main ():

final circle = shapeFactory('circle');

final square = shapeFactory('square');

Ta chạy và output sẽ giống như lúc trước

Option 2: Tạo một factory constructor. Thêm một constructor trong abstract class Shape

abstract class Shape {

factory Shape(String type) {

if (type == 'circle') return Circle(2);

if (type == 'square') return Square(2);

throw 'Can\'t create $type.';

}

num get area;

}

Thay thế hai dòng đầu tiên của main () bằng đoạn mã sau:

final circle = Shape('circle');

final square = Shape('square');

Xóa 2 hàm shapeFactory () mà ta đã thêm trước đó. sau đó chạy thì kết quả vẫn như 2 lần trước.

**Implement một interface**

Dart không có từ khoá **interface** bởi ***every class defines an interface.***

Thêm một lớp CircleMock và implement Circle interface của mục 4. Và ta bắt buộc phải xác định các biến thể hiện area và radius

class CircleMock implements Circle {

num area = 0;

num radius = 0;

}

**Sử dụng Dart cho functional programming**

Trong functional programming, ta có thể làm những việc sau:

Truyền functions (hàm hay phương thức) như một tham số

Gán một function cho một biến.

Phân rã cấu trúc một hàm có nhiều tham số thành nhiều hàm mà mỗi hàm nhận mội tham số (gọi là currying)

Tạo một hàm không tên có thể được sử dụng như một giá trị không đổi (còn được gọi là biểu thức lambda; biểu thức lambda đã được thêm vào Java trong bản phát hành JDK 8).

Dart hỗ trợ tất cả các tính năng đó. Trong Dart, các hàm là các object và có một kiểu, Function. Điều này có nghĩa là các hàm có thể được gán cho các biến hoặc được chuyển làm đối số cho các hàm khác. Ta cũng có thể gọi một thể hiện của lớp Dart như thể nó là một hàm, như ví dụ:

String scream(int length) => "A${'a' \* length}h!";

main() {

final values = [1, 2, 3, 5, 10, 50];

for (var length in values) {

print(scream(length));

}

}

output:

Aah!

Aaah!

Aaaah!

Aaaaaah!

Aaaaaaaaaaah!

Aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaah!

Khi sử dụng phép nội suy chuỗi, chuỗi $ {'a' \* length} được đánh giá là "ký tự 'a' lặp lại length lần.

Tiếp theo ta loại bỏ vòng lặp for () {...} bắt buộc trong main () và thay thế nó bằng một dòng mã sử dụng chuỗi phương thức:

values.skip(1).take(3).map(scream).forEach(print);

Chạy thì output sẽ là:

Aaah!

Aaaah!

Aaaaaah!

skip(1) có nghĩa là loại bỏ giá trị đầu tiên, take(3) là chọn ra 3 giá trị tiếp theo, ở đây là 2, 3 và 5. Các giá trị còn lại bỏ qua.

1. **CÀI ĐẶT THỰC NGHIỆM**
2. **Cài đặt triển khai**

Để phát triển ứng dụng chatapp sử dụng Flutter và Dart, và kết nối với server thông qua giao thức WebSocket và thư viện SocketIO, cùng với việc lấy dữ liệu từ server bằng cách gọi API sử dụng networking, HTTP và JSON, bạn có thể làm theo các bước sau:

Bước 1: Chuẩn bị môi trường phát triển

- Cài đặt Flutter SDK và cấu hình môi trường để phát triển ứng dụng Flutter trên máy tính của bạn.

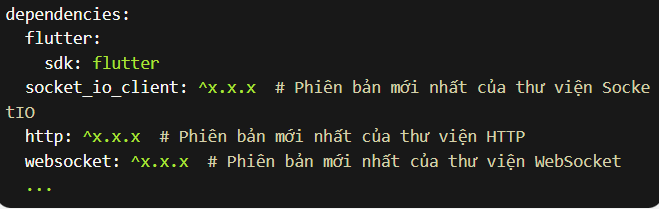
Bước 2: Tạo một ứng dụng Flutter mới

- Mở terminal hoặc command prompt và chạy lệnh sau để tạo một ứng dụng Flutter mới:

Flutter create chat\_app

Bước 3: Thêm các dependencies cần thiết

- Mở file `pubspec.yaml` trong thư mục gốc của dự án Flutter và thêm dependencies sau:



Hình 3. 1 Thêm dependencies

- Chạy lệnh sau để cập nhật dependencies:

Flutter pub get

Bước 4: Xử lý kết nối WebSocket và SocketIO

- Trong mã nguồn Flutter, tạo một kết nối WebSocket hoặc SocketIO đến server. Bạn có thể sử dụng thư viện `socket\_io\_client` để thực hiện điều này. Dưới đây là một ví dụ về cách kết nối đến server sử dụng SocketIO:

A computer screen with text

Description automatically generated

Hình 3. 2 Ví dụ về cách kết nối đến server sử dụng SocketIO

- Bạn cũng có thể sử dụng giao thức WebSocket trực tiếp thay vì sử dụng thư viện SocketIO. Để làm điều này, bạn có thể sử dụng thư viện `websocket`. Dưới đây là một ví dụ về cách kết nối đến server sử dụng WebSocket:

A screen shot of a computer program

Description automatically generated

Hình 3. 3 ví dụ về cách kết nối đến server sử dụng WebSocket

Bước 5: Lấy dữ liệu từ server bằng cách gọi API

- Sử dụng thư viện `http` để gửi yêu cầu HTTP đến server và lấy dữ liệu từ API. Dưới đây là một ví dụ về cách lấy dữ liệu từ server:

A computer screen shot of a code

Description automatically generated

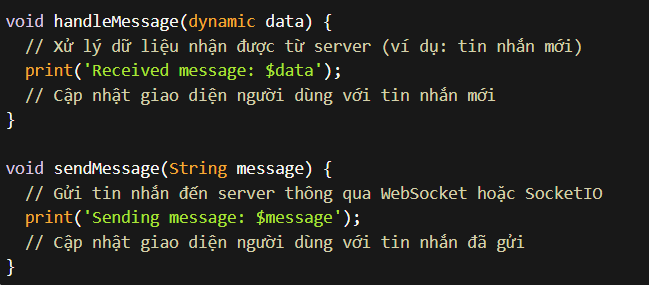
Hình 3. 4 Ví dụ về cách lấy dữ liệu từ server

Bước 6: Xây dựng giao diện người dùng

- Sử dụng Flutter để xây dựng giao diện người dùng cho ứng dụng chatapp. Bạn có thể sử dụng các widget như `ListView`, `TextField`, `FlatButton`, và `StreamBuilder` để hiển thị danh sách tinnhắn, trường nhập tin nhắn và các nút để gửi tin nhắn.

Bước 7: Xử lý dữ liệu nhận được và gửi đi

- Khi nhận được dữ liệu từ server thông qua WebSocket hoặc SocketIO, hoặc khi người dùng gửi tin nhắn, bạn có thể xử lý dữ liệu và thực hiện các tương tác tương ứng trên giao diện người dùng. Ví dụ:



Hình 3. 5 Ví dụ về xử lý dữ liệu và thực hiện các tương tác

Bước 8: Đóng kết nối và giải phóng tài nguyên

- Khi ứng dụng không cần thiết sử dụng kết nối WebSocket hoặc SocketIO nữa, hãy đảm bảo rằng bạn đóng kết nối và giải phóng tài nguyên để tránh lãng phí tài nguyên của hệ thống.

A black screen with white text

Description automatically generated

Hình 3. 6 Đóng kết nối và giải phóng tài nguyên

Đây chỉ là một hướng dẫn cơ bản để phát triển ứng dụng chatapp sử dụng Flutter, Dart, WebSocket, SocketIO, và các thư viện HTTP, JSON. Bạn có thể điều chỉnh và mở rộng dự án theo nhu cầu cụ thể của mình.

1. **Thiết kế cơ sở dữ liệu**

A diagram of a conversation and user connection

Description automatically generated with medium confidenceHình 3. 7 ERD

A screenshot of a computer

Description automatically generated

Hình 3. 8 Bảng message

A screenshot of a computer

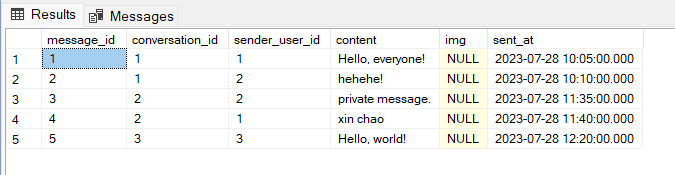
Description automatically generated

Hình 3. 9 Bảng conversation

A screenshot of a computer

Description automatically generated

Hình 3. 10 Bảng Users



Hình 3. 11 Cơ sở dữ liệu bản Message

A screenshot of a computer

Description automatically generated

Hình 3. 12 Cơ sở dữ liệu bảng User

A screenshot of a computer

Description automatically generated

Hình 3. 13 Cơ sở dữ liệu bảng conversation

A diagram of a server

Description automatically generated

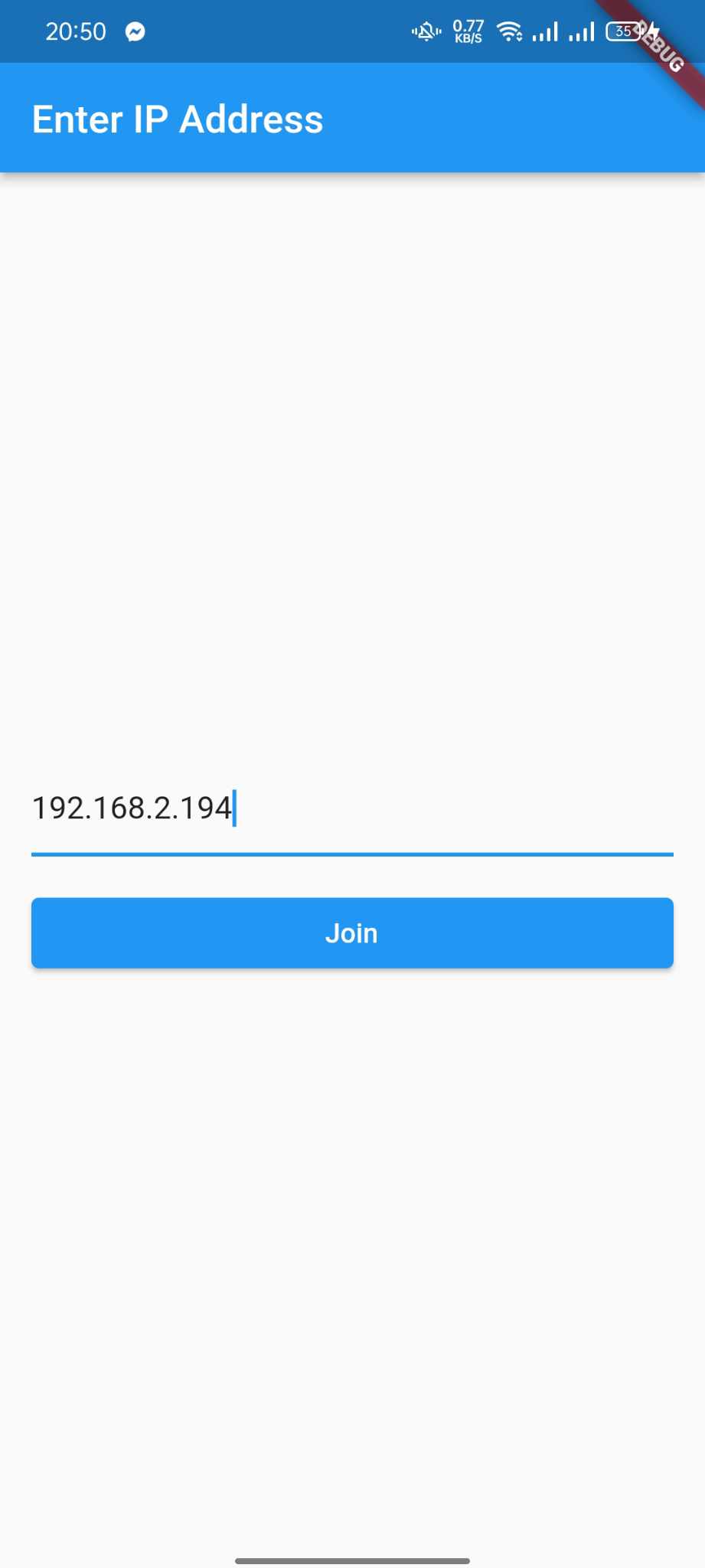
Hình 3. 14 call api http

A diagram of a server web schema

Description automatically generated

Hình 3. 15 Chat real time websocke

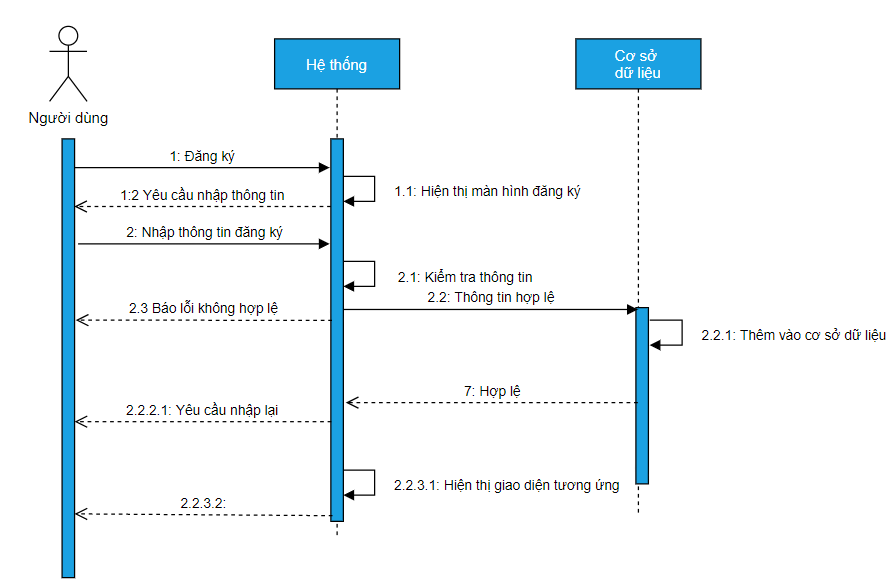
1. **Thiết kế các chức năng của app**



Hình 3. 16 Địa chỉ IP

Truy cập vào địa chỉ IP của máy server để sử dụng app chat

Khi nhập địa chỉ IP vào, appchat sẽ gửi yêu cầu đến địa chỉ IP đó và thiết lập kết nối với máy chủ tương ứng. Sau đó, máy chủ có thể phản hồi bằng việc cung cấp các dịch vụ của appchat

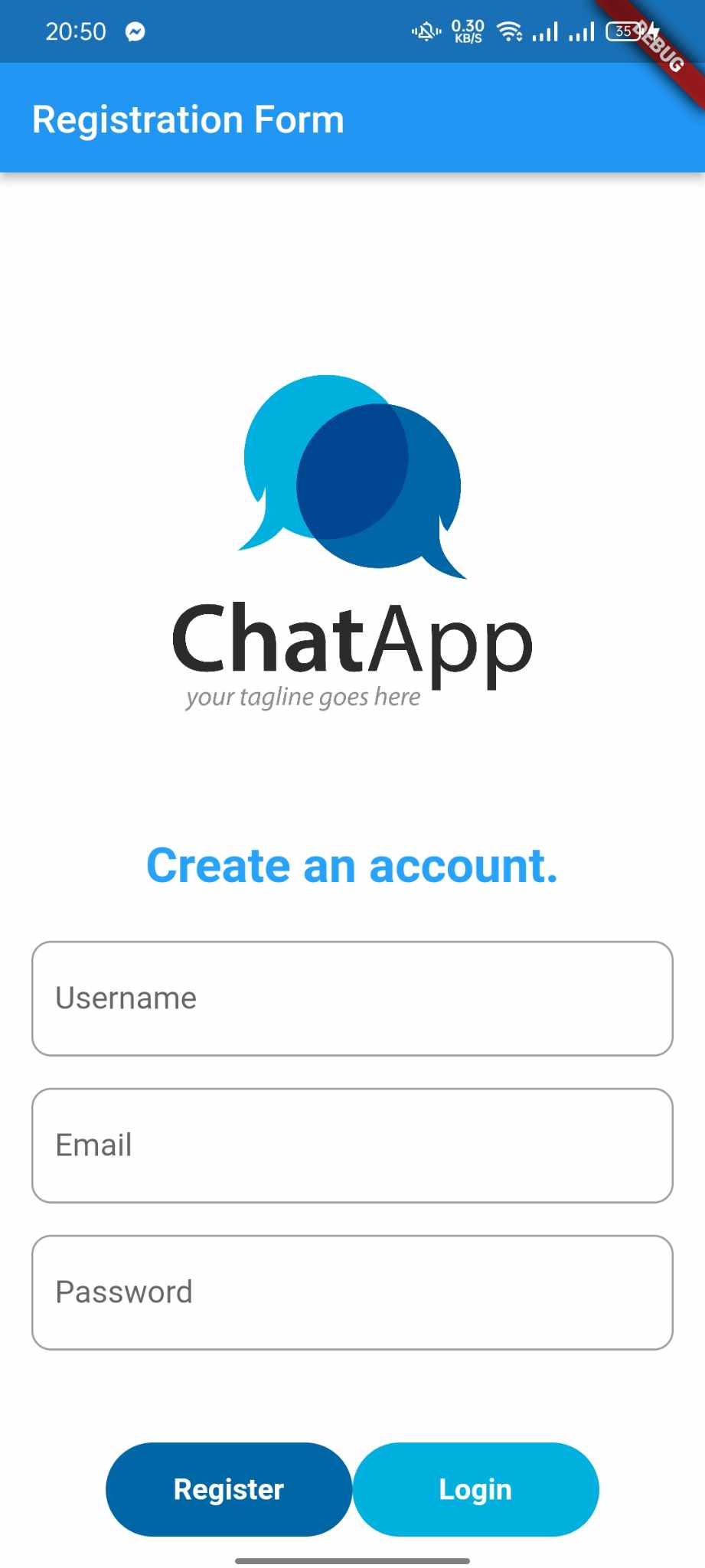


Hình 3. 17 Sơ đồ tương tác của chức năng đăng ký

A diagram of a diagram

Description automatically generated

Hình 3. 18 Sơ đồ hoạt động của chức năng đăng ký



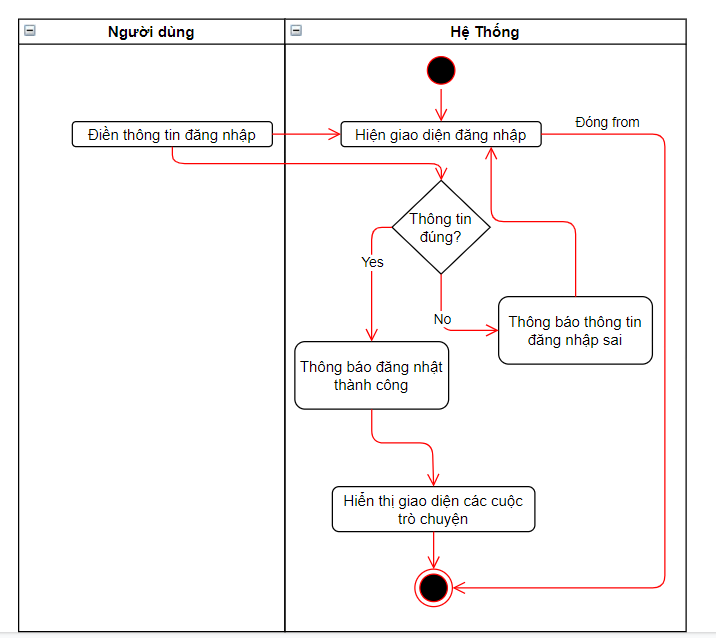
Hình 3. 19 Giao diện đăng ký

Chức năng đăng ký là một phần quan trọng của hệ thống và ứng dụng chat. Nó cho phép người dùng tạo tài khoản cá nhân để truy cập và sử dụng các tính năng và dịch vụ của hệ thống. Dưới đây là một sơ lược về chức năng đăng ký:

Giao diện đăng ký: Người dùng truy cập vào giao diện đăng ký. Giao diện đăng ký yêu cầu người dùng cung cấp một số thông tin cá nhân cần thiết để tạo tài khoản.

Thông tin cá nhân: Người dùng cung cấp thông tin cá nhân như tên, địa chỉ email và mật khẩu. Thông tin này được sử dụng để xác thực và bảo mật tài khoản của người dùng.

Tạo tài khoản: Sau khi thông tin cá nhân được xác thực, hệ thống sẽ tạo tài khoản cho người dùng. Tài khoản này sẽ được liên kết với thông tin cá nhân và được sử dụng để đăng nhập vào hệ thống sau này

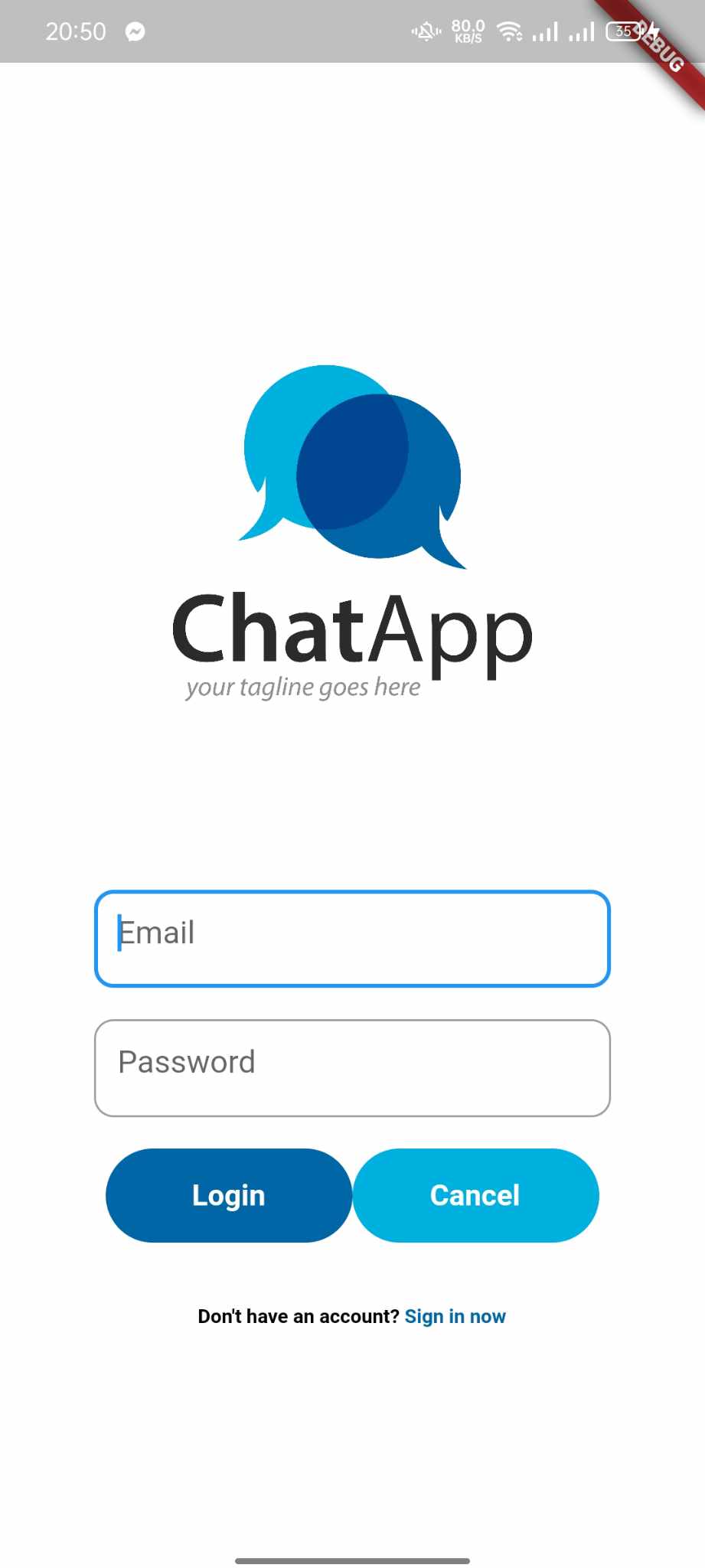


Hình 3. 20 Mô hình hoạt động của chức năng đăng nhập

A diagram of a diagram

Description automatically generated

Hình 3. 21 Mô hình tương tác của chức năng đăng nhập



Hình 3. 22 Giao diện đăng nhập

Chức năng đăng nhập là một quy trình cho phép người dùng xác thực và truy cập vào tài khoản cá nhân của mình trên ứng dụng. Nó đảm bảo rằng chỉ những người dùng có thông tin đăng nhập chính xác mới có thể truy cập vào các tính năng, dịch vụ và thông tin cá nhân.

Quá trình đăng nhập diễn ra như sau:

Nhập thông tin đăng nhập: Người dùng cung cấp thông tin đăng nhập, là tên đăng nhập (hoặc địa chỉ email) và mật khẩu, vào giao diện đăng nhập của ứng dụng.

Xác thực thông tin: Hệ thống sẽ xác thực thông tin đăng nhập được cung cấp bằng cách so sánh với dữ liệu đã lưu trữ. Nếu thông tin đúng, người dùng được cho phép tiếp tục vào hệ thống.

Truy cập vào tài khoản: Sau khi xác thực thành công, người dùng có quyền truy cập vào tài khoản cá nhân của mình. Họ có thể sử dụng các tính năng, dịch vụ và tài nguyên được cung cấp bởi hệ thống.

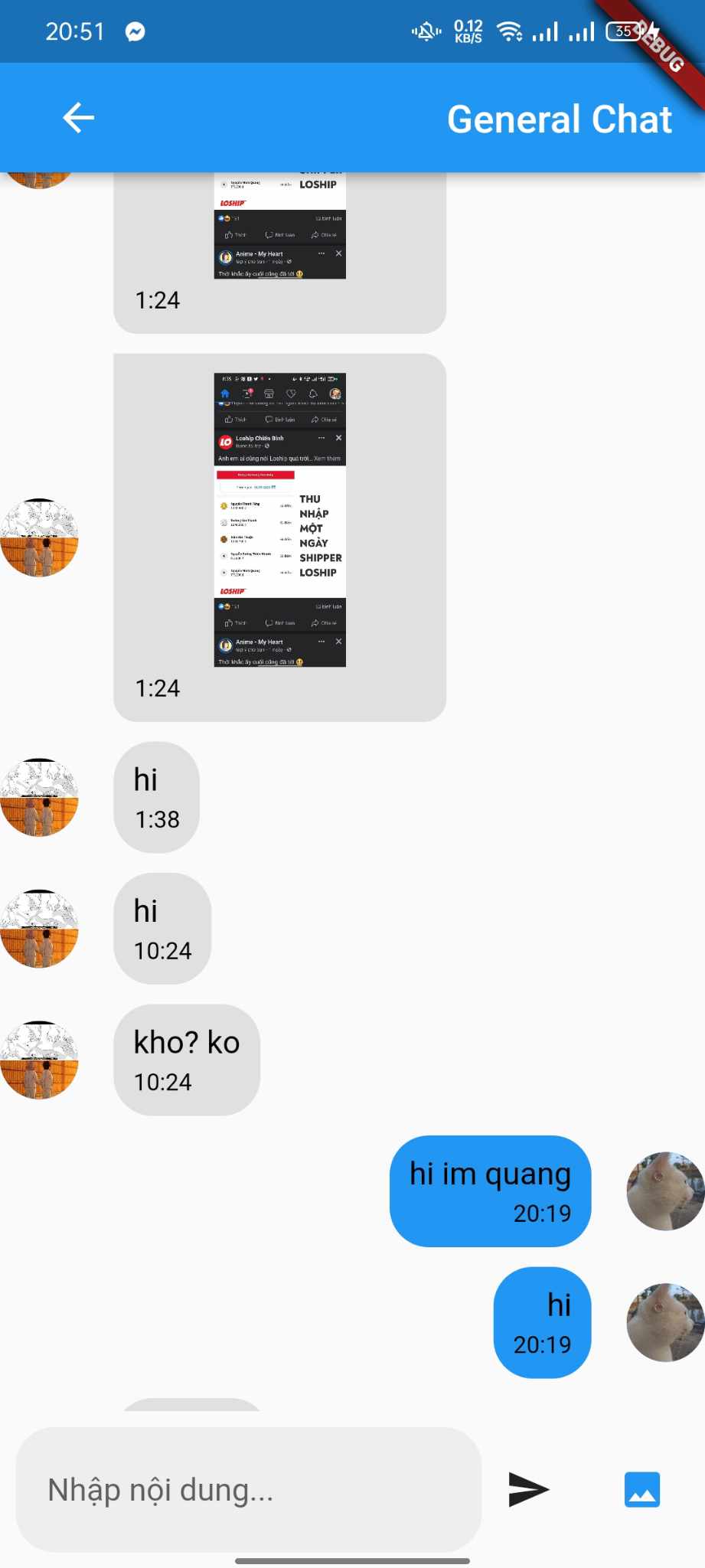
Quản lý tài khoản: Người dùng có thể quản lý thông tin cá nhân, cập nhật cài đặt và thực hiện các hành động khác liên quan đến tài khoản của mình.

Chức năng đăng nhập giúp đảm bảo tính bảo mật và riêng tư cho người dùng, vì chỉ những người dùng có thông tin đăng nhập chính xác mới có thể truy cập vào các tài nguyên và thông tin cá nhân. Nó cũng cho phép hệ thống tạo ra trải nghiệm cá nhân hóa và cung cấp quyền truy cập chỉ cho những người dùng đã được xác thực.

A diagram of a diagram

Description automatically generated with medium confidence

Hình 3. 23 Mô hình hoạt động của chức năng trò chuyện



Hình 3. 24 Giao diện cuộc trò chuyện

Cuộc trò chuyện là một hoạt động giao tiếp giữa hai người nhằm trao đổi ý kiến, thông tin và ý tưởng. Nó có thể diễn ra trong ứng dụng trò chuyện trực tuyến.

Giao diện trò chuyện là phần mềm hoặc môi trường mà người dùng sử dụng để tham gia vào các hoạt động trò chuyện, giao tiếp với nhau. Giao diện trò chuyện thường được thiết kế để tạo ra một trải nghiệm dễ sử dụng và tương tác thông qua các thành phần và chức năng sau:

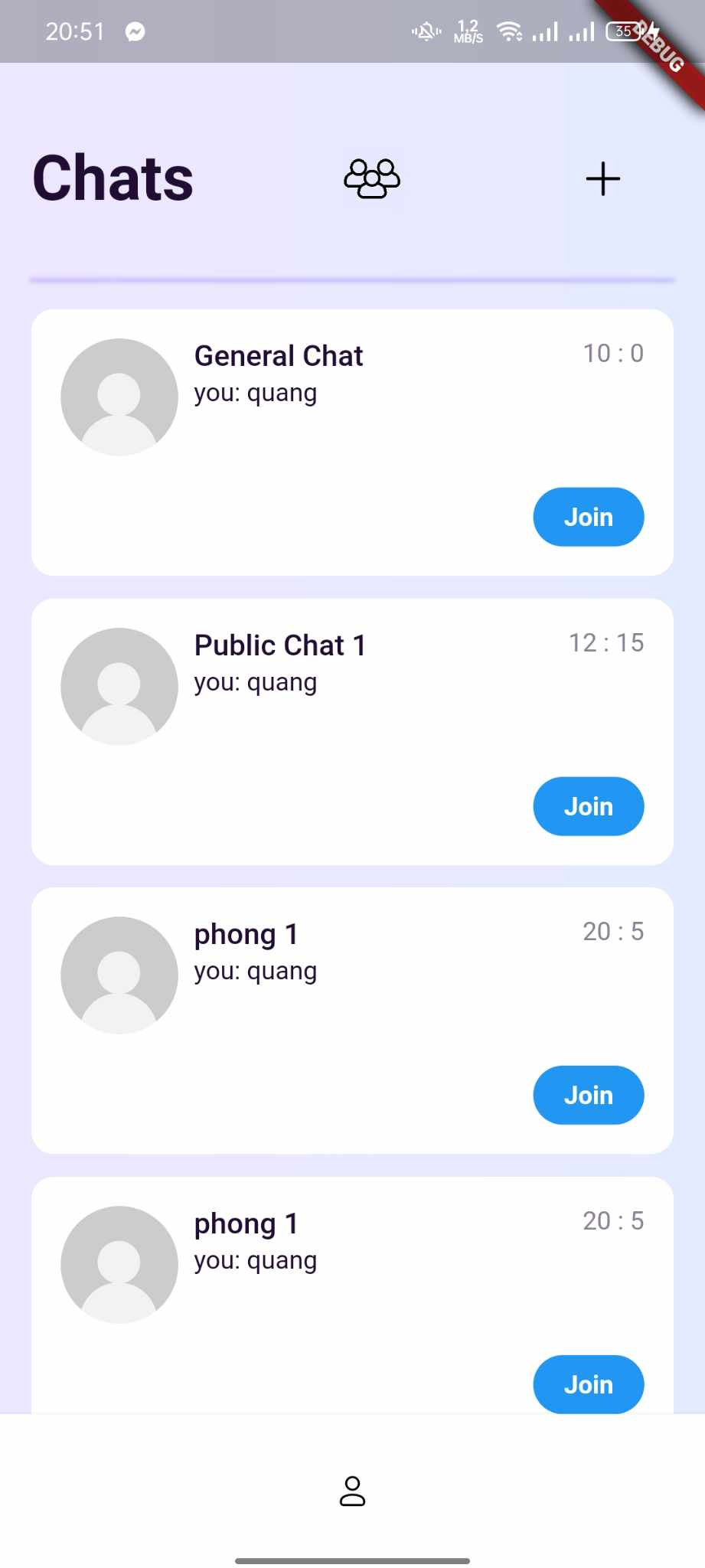
1. Cửa sổ chat: Đây là phần chính trong giao diện trò chuyện, nơi người dùng có thể nhập và hiển thị tin nhắn. Các tin nhắn mới được hiển thị ở cuối cùng hoặc trên cùng của cửa sổ, và người dùng có thể cuộn lên hoặc xuống để xem tin nhắn cũ hơn.

2. Khung nhập tin nhắn: Đây là nơi người dùng nhập nội dung tin nhắn để gửi đi. Nó thường được đặt ở dưới cùng hoặc ở một bên cạnh của cửa sổ chat. Có thể có các biểu tượng hoặc nút gửi để gửi tin nhắn.

3. Biểu tượng cảm xúc: Giao diện trò chuyện thường cung cấp các biểu tượng cảm xúc để người dùng thể hiện tình cảm hoặc phản ứng của họ. Các biểu tượng này có thể là hình mặt cười, biểu tượng tương tác hoặc biểu tượng tùy chỉnh.

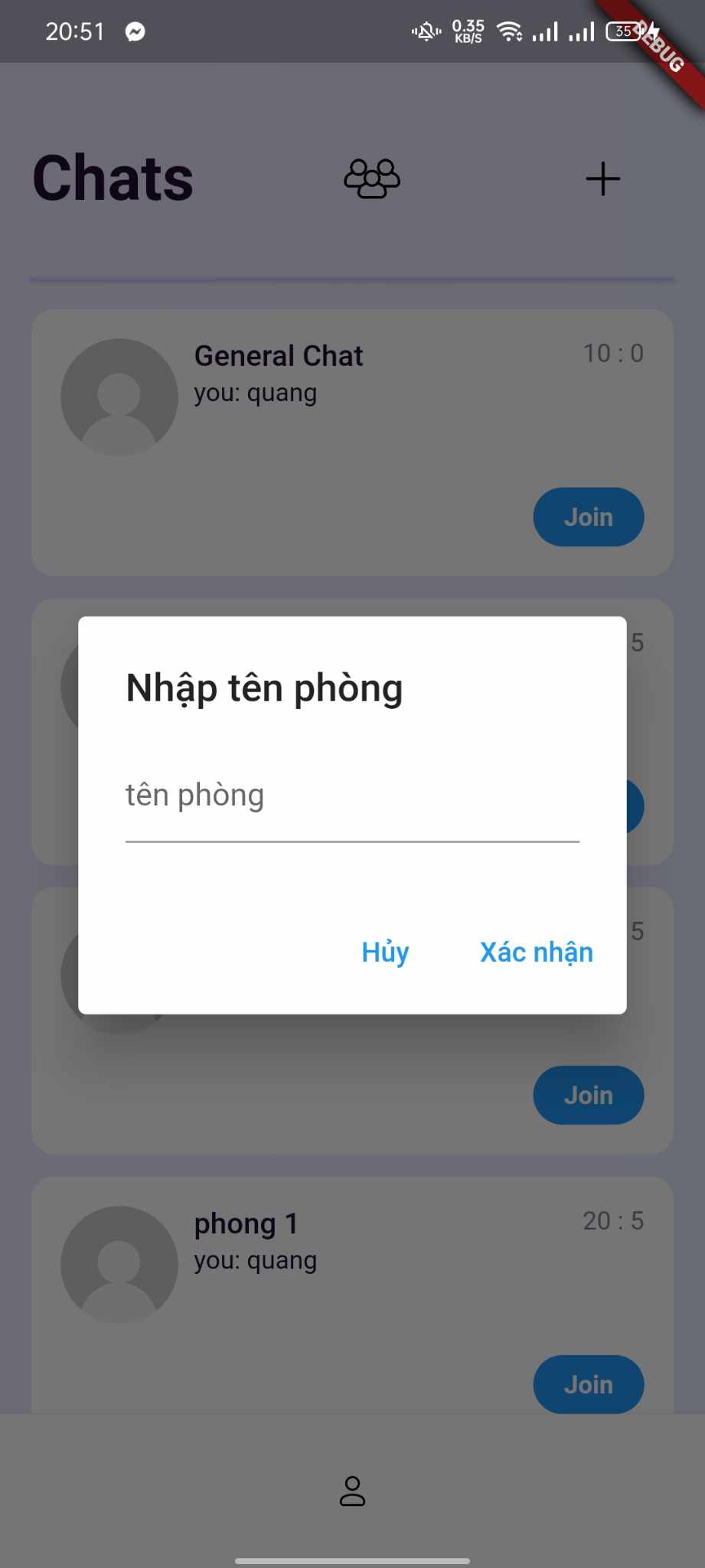
4. Hỗ trợ đa phương tiện: Một giao diện trò chuyện có thể hỗ trợ việc chia sẻ đa phương tiện như hình ảnh. Các chức năng này cho phép người dùng chia sẻ nội dung trực quan và nâng cao trải nghiệm trò chuyện.

Giao diện trò chuyện thường được thiết kế để đơn giản, trực quan và dễ sử dụng. Nó tập trung vào việc cung cấp các công cụ và chức năng cần thiết để người dùng có thể dễ dàng trò chuyện và tương tác với nhau trong một môi trường trực tuyến.



Hình 3. 25 Giao diện danh sách người dùng

Danh sách người dùng: Giao diện trò chuyện thường hiển thị danh sách người dùng đang tham gia cuộc trò chuyện. Điều này cho phép người dùng biết ai đang có mặt trong phòng chat và tương tác với họ thông qua tin nhắn riêng hoặc các hoạt động khác.

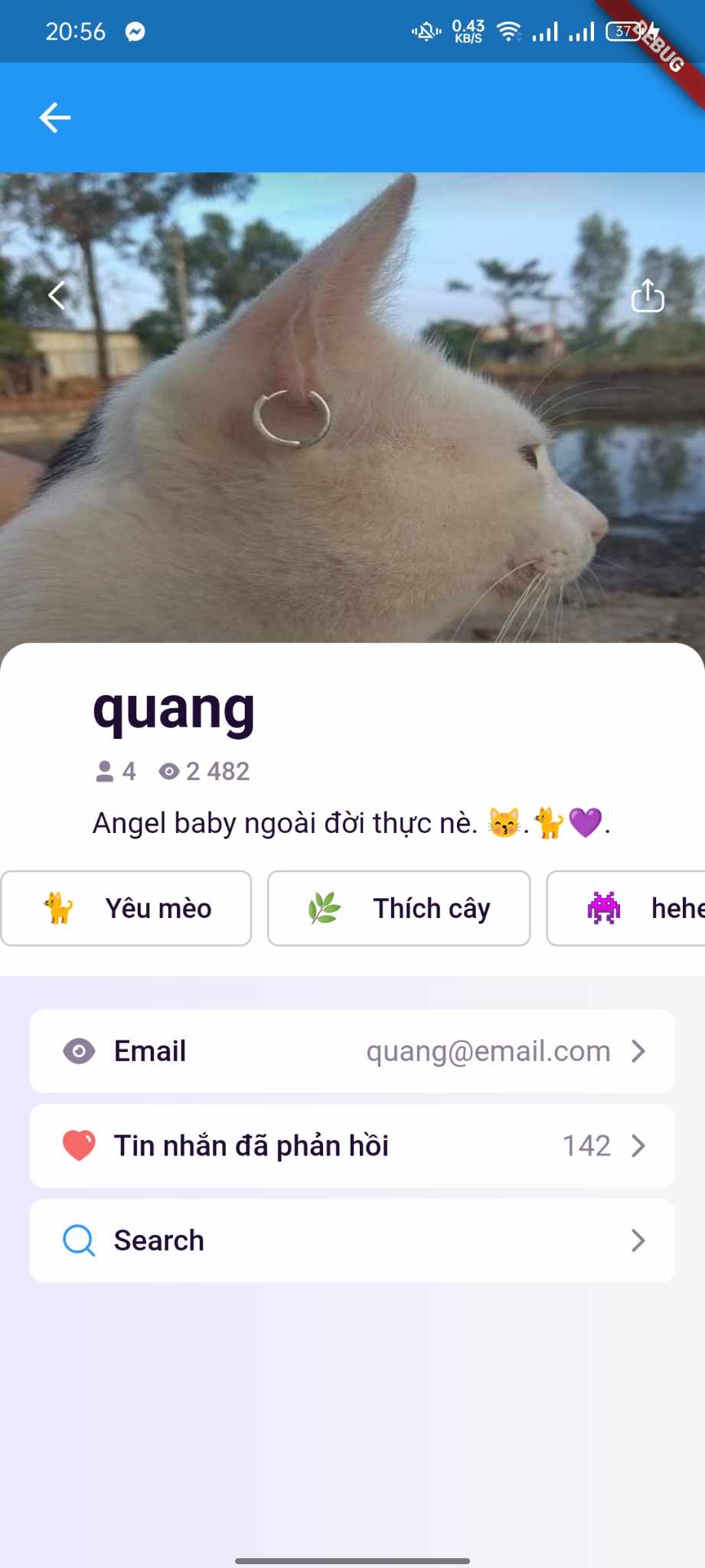


Hình 3. 26 Giao diện tạo phòng chat nhóm

Giao diện tạo phòng chat được thiết kế để hỗ trợ người dùng tạo ra các phòng chat mới trên nền tảng trò chuyện nhóm.

Tên phòng chat: Người dùng có thể nhập tên cho phòng chat mới mà họ định tạo. Tên này thường được sử dụng để định danh và phân biệt phòng chat với các phòng chat khác.

Thành viên phòng chat: Giao diện tạo phòng chat thường cung cấp một cách để người dùng quản lý thành viên trong phòng chat. Điều này có thể bao gồm mời thành viên bằng cách nhập tên, hoặc chọn từ danh sách các thành viên hiện có.



Hình 3. 27 Giao diện trang cá nhân

Giao diện trang cá nhân là một phần trong một ứng dụng mạng xã hội nơi người dùng có thể quản lý và hiển thị thông tin cá nhân của mình. Dưới đây là sơ lược về giao diện trang cá nhân:

1. Thông tin cá nhân: Giao diện trang cá nhân thường hiển thị các thông tin cá nhân của người dùng, bao gồm tên, ảnh đại diện, tiểu sử, thông tin liên hệ và các mục khác liên quan đến cá nhân của người dùng. Giao diện này cho phép người dùng cập nhật và chỉnh sửa thông tin cá nhân của mình.

2. Ảnh đại diện: Giao diện thường cung cấp một cách để người dùng tải lên hoặc chọn ảnh đại diện. Người dùng có thể tải lên ảnh từ máy tính hoặc chọn ảnh từ các album ảnh có sẵn.

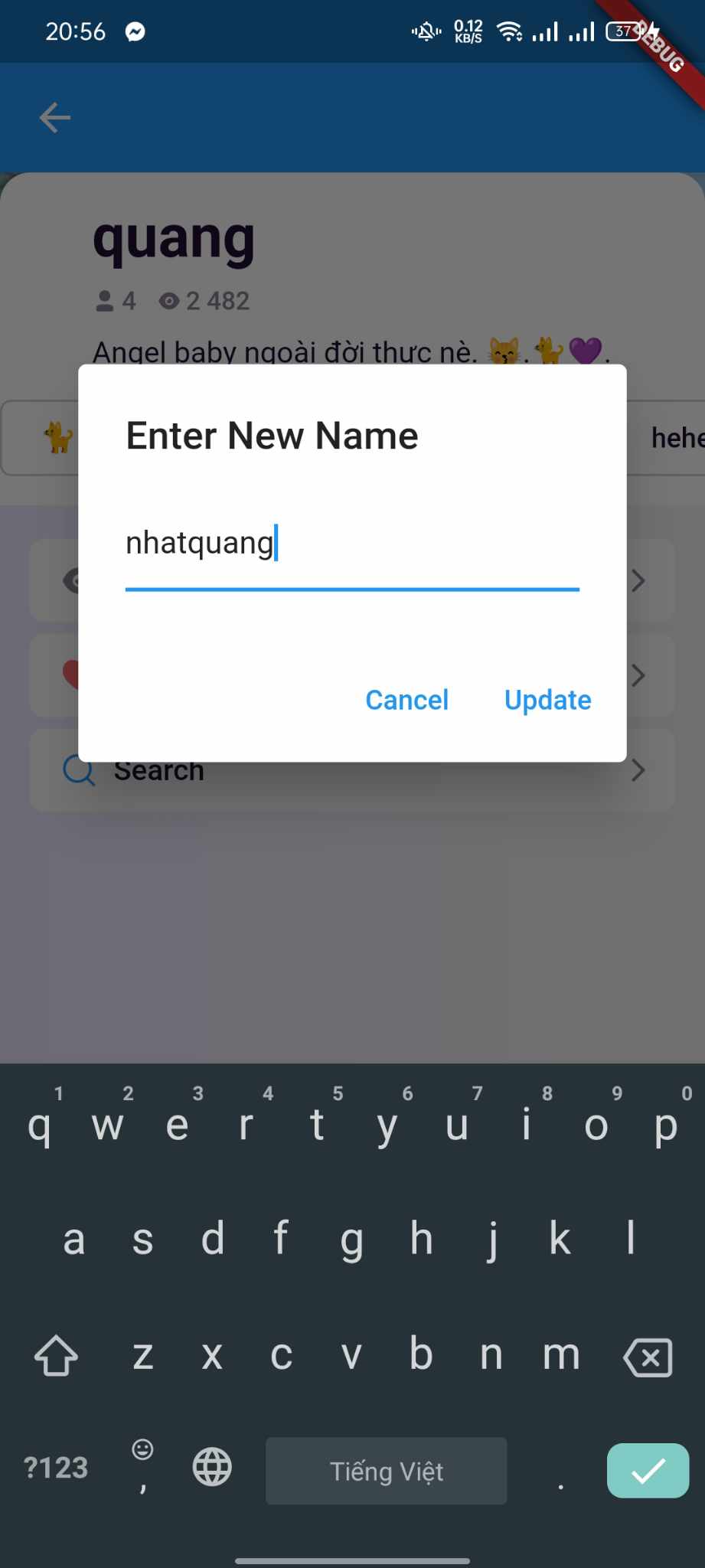
3. Bài đăng hoặc trạng thái: Giao diện trang cá nhân thường cho phép người dùng viết và đăng các bài đăng hoặc trạng thái. Người dùng có thể chia sẻ thông tin, suy nghĩ, hình ảnh, video hoặc các nội dung khác với người khác thông qua bài đăng hoặc trạng thái.

4. Quản lý và hiển thị danh sách bạn bè: Giao diện thường cung cấp một phần để người dùng quản lý và hiển thị danh sách bạn bè của họ. Người dùng có thể thêm bạn bè mới, chấp nhận hoặc từ chối lời mời kết bạn, xem danh sách bạn bè hiện tại và thực hiện các hoạt động khác liên quan đến quản lý bạn bè.

5. Cài đặt riêng tư: Giao diện trang cá nhân thường cung cấp các tùy chọn cài đặt riêng tư. Người dùng có thể kiểm soát ai có thể xem thông tin cá nhân, bài đăng, ảnh đại diện và các hoạt động khác trên trang cá nhân của mình.

6. Các tab hoặc phần trên trang cá nhân: Giao diện thường chia trang cá nhân thành các tab hoặc phần khác nhau để người dùng có thể xem các thông tin cụ thể như bài viết, ảnh, video, sự kiện hoặc các thông tin khác.

Giao diện trang cá nhân được thiết kế để cho phép người dùng quản lý và hiển thị thông tin cá nhân, chia sẻ nội dung và tương tác với người khác. Nó cung cấp các phần để người dùng cập nhật thông tin cá nhân, tải lên ảnh đại diện, viết bài đăng hoặc trạng thái, quản lý danh sách bạn bè và cài đặt quyền riêng tư.



Hình 3. 28 Giao diện đổi tên người dùng

Giao diện đổi tên người dùng là một phần trong hệ thống hoặc ứng dụng nơi người dùng có thể thay đổi tên hiển thị của họ. Dưới đây là một sơ lược về giao diện đổi tên người dùng:

Hiển thị tên hiện tại: Giao diện đổi tên người dùng thường hiển thị tên hiện tại của người dùng, cho phép họ xem và kiểm tra thông tin hiện tại trước khi thay đổi.

Ô nhập tên mới: Giao diện cung cấp một ô nhập liệu hoặc vùng văn bản cho người dùng nhập tên mới. Người dùng có thể gõ tên mới vào ô nhập liệu này.

Xác nhận thay đổi: Giao diện thường có một nút hoặc tùy chọn để người dùng xác nhận thay đổi tên. Khi người dùng nhấp vào nút này, tên mới sẽ được áp dụng và cập nhật trên hệ thống.

1. **KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN**

## **Kết luận**

Trong đề tài này, chúng ta đã làm được những điều sau đây:

1. Tìm hiểu về cách xử lý kết nối và truyền tải dữ liệu real-time giữa client và server sử dụng giao thức websocket và thư viện SocketIO.

2. Tìm hiểu về việc lấy dữ liệu từ server bằng cách call api sử dụng networking & HTTP, JSON.

3. Thiết kế và xây dựng một ứng dụng chatapp sử dụng các công nghệ trên để truyền tải dữ liệu real-time và cung cấp các tính năng chat, gửi tệp tin và ảnh, đăng nhập và đăng ký tài khoản.

Tuy nhiên, đề tài này vẫn còn một số nhược điểm và hạn chế, bao gồm:

1. Đề tài chưa đi sâu vào các chi tiết kỹ thuật của các công nghệ sử dụng trong đề tài, nên các nhà phát triển mới bắt đầu tìm hiểu về chúng có thể gặp khó khăn.

2. Ứng dụng chatapp được thiết kế và xây dựng trong đề tài vẫn chưa đầy đủ các tính năng và chức năng như một ứng dụng chatapp thực tế.

3. Các kỹ thuật và công nghệ được sử dụng trong đề tài có thể bị lỗi hoặc không tương thích với một số thiết bị hoặc phiên bản phần mềm khác nhau.

Tổng kết lại, đề tài này đã đem lại nhiều kiến thức và kinh nghiệm quý giá về cách xử lý kết nối và truyền tải dữ liệu real-time giữa client và server sử dụng giao thức websocket và thư viện SocketIO, cùng với việc lấy dữ liệu từ server bằng cách call api sử dụng networking & HTTP, JSON để phát triển ứng dụng chatapp bằng Flutter và Dart. Tuy nhiên, đề tài vẫn còn những hạn chế và cần được phát triển và hoàn thiện hơn để đáp ứng được nhu cầu của người dùng.

## **Hướng phát triển**

Để phát triển ứng dụng chatapp hiệu quả, cần tập trung vào thiết kế và xây dựng các tính năng đáp ứng nhu cầu của người dùng, đồng thời đảm bảo tính bảo mật và hiệu năng của ứng dụng. Các hướng phát triển tiếp theo có thể bao gồm:

Tích hợp các tính năng mới như gọi thoại và gọi video.

Tối ưu hoá hiệu năng của ứng dụng để đảm bảo truyền tải dữ liệu real-time mượt mà và ổn định.

Tăng tính bảo mật của ứng dụng bằng cách sử dụng mã hóa dữ liệu và xác thực người dùng.

Tích hợp tính năng chatbot để tăng tính tương tác và giảm tải cho người dùng.

Phát triển ứng dụng cho các nền tảng khác như web và desktop để phục vụ cho nhu cầu sử dụng của đa dạng người dùng.

Với sự phát triển không ngừng của công nghệ và nhu cầu sử dụng ứng dụng chatapp ngày càng cao, đề tài này có tiềm năng phát triển và ứng dụng rộng rãi trong tương lai.

# **TÀI LIỆU THAM KHẢO**

1. Al-Jarrah, O. Y., & Al-Khasawneh, A. (2019). Performance Evaluation of WebSockets and HTTP/2 for Real-Time Web Applications. International Journal of Computer Science and Network Security, 19(2), 152-157.
2. Dart Programming Language. (n.d.). Retrieved
3. Flutter - Beautiful native apps in record time. (n.d.). Retrieved
4. Flutter HTTP Package. (n.d.). Retrieved
5. Flutter JSON Serialization and Deserialization. (n.d.). Retrieved
6. FlutterWebSocket Package. (n.d.). Retrieved
7. Obrst, L., Ceusters, W., & Mani, I. (2012). Ontologies and the Lexicon. In Handbook of Ontologies (pp. 319-342). Springer.
8. Rosenberg, J. (2015). Websockets vs. REST: Understanding the Difference. Retrieved
9. SockJS. (n.d.). Retrieved
10. WebSockets API - W3C Recommendation. (2011). Retrieved