**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 CUỐI KÌ**

**MÔN: CÔNG NGHỆ LẬP TRÌNH ĐA NỀN TẢNG**

**CHO ỨNG DỤNG DI ĐỘNG**

**ĐỀ TÀI: XÂY DỰNG APP CHAT**

**BẰNG FLUTTER VÀ SOCKET.IO**

****

Giảng viên hướng dẫn : **ThS. Nguyễn Thanh Truyền**

Sinh viên thực hiện: **Văn Trung Nghĩa 0850080085**

**Nguyễn Phi Trường 0850080110**

**Nguyễn Minh Quang 0850080099**

**Ngô Phúc Thịnh 0850080106**

**Nguyễn Tiến Bách 0850080060**

**Hoàng Quang Tuấn 0850080112**

Lớp: **08\_ĐH\_CNPM**

Khoá: **08**

***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 CUỐI KÌ**

**MÔN: CÔNG NGHỆ LẬP TRÌNH ĐA NỀN TẢNG**

**CHO ỨNG DỤNG DI ĐỘNG**

**ĐỀ TÀI: XÂY DỰNG APP CHAT**

**BẰNG FLUTTER VÀ SOCKET.IO**

****

Giảng viên hướng dẫn : **ThS. Nguyễn Thanh Truyền**

Sinh viên thực hiện: **Văn Trung Nghĩa 0850080085**

**Nguyễn Phi Trường 0850080110**

**Nguyễn Minh Quang 0850080099**

**Ngô Phúc Thịnh 0850080106**

**Nguyễn Tiến Bách 0850080060**

**Hoàng Quang Tuấn 0850080112**

Lớp: **08\_ĐH\_CNPM**

Khoá: **08**

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

**MỞ ĐẦU**

Trong thời đại kỹ thuật số hiện nay, ứng dụng di động đã trở thành một phần không thể thiếu trong cuộc sống hàng ngày của chúng ta. Cùng với sự phát triển của công nghệ, việc giao tiếp và kết nối giữa con người ngày càng dễ dàng và tiện lợi hơn. Trong bối cảnh đó, ứng dụng chat đóng vai trò cực kỳ quan trọng và trở thành một trong những ứng dụng phổ biến nhất trên các nền tảng di động. Từ việc gửi tin nhắn văn bản đơn giản cho đến cuộc trò chuyện thời gian thực với hình ảnh và âm thanh, ứng dụng chat đã mang lại sự kết nối mạnh mẽ cho hàng tỷ người trên toàn cầu.

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

Qua đó mục tiêu đáp ứng nhu cầu ngày càng cao về ứng dụng chat đáng tin cậy, an toàn và hiệu quả, nhóm chúng em đã chọn đề tài **“Xây dựng app chat bằng Flutter và Socket.io”.** Đề này tập trung vào việc xây dựng một ứng dụng chat sử dụng hai công nghệ đột phá: Flutter và Socket.io

**LỜI CẢM ƠN**

Trong quá trình thực hiện 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 chúng em. Những kiến thức này sẽ rất hữu ích cho chúng em trong công việc phát triển ứng dụng di động trong tương lai.

Mặc dù đã rất cố gắng nhưng do thời gian có hạn và kiến thức còn hạn chế nên trong quá trình làm đồ án không thể tránh khỏi những thiếu sót. Chính vì vậy em mong thầy và các bạn góp ý để đồ án của chúng em có thể hoàn thiện tốt hơn.

Chúng em xin chân thành cảm ơn!

**NHẬN XÉT**

**(Của giảng viên)**

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

………….., ngày….tháng….năm……

NGƯỜI NHẬN XÉT

*(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 Phi Trường** | - Code (ngôn ngữ dart, framework fluter, IDE:Android studio, Server: NodeJS)  - Code chung với Nghĩa:  + Code giao diện xác thực, chọn vùng quốc gia …  + Code các chức năng để thực hiện việc call api lên server HTTP được viết bằng NodeJS để lấy dữ liệu.  + Deploy nodejs server lên vercel  - Lý thuyết async, Socket.io |  |
| **Văn Trung Nghĩa** | - Code (ngôn ngữ dart, framework fluter, IDE:Visual studio, Server: NodeJS)  - Code chung với Trường:  + Code Server bằng NodeJS  + Code các giao diện phòng chat, chuyển màn hình, camera … và xử lý các chức năng khác  - Lý thuyết future, Socket.io |  |
| **Nguyễn Minh Quang** | + Lý thuyết await  + Làm PPT. |  |
| **Nguyễn Tiến Bách** | + Code các giao diện code chung với nghĩa  + Test dữ liệu trả về thông qua phần mềm postman  + Lý thuyết expanded, Call http request |  |
| **Ngô Phúc Thịnh** | + Lý Thuyết Flexible  + Làm word với Tuấn |  |
| **Hoàng Quang Tuấn** | + Lý Thuyết Stack Widget  + Làm word với Thịnh |  |

TP. Hồ Chí Minh, ngày…tháng….năm 2023

**Sinh Viên**

**MỤC LỤC**

[CHƯƠNG 1. TỔNG QUAN 1](#_Toc142421491)

[1.1 Lý do chọn đề tài 1](#_Toc142421492)

[1.2 Phạm vi đề tài 1](#_Toc142421493)

[1.3 Mục tiêu và nhiệm vụ 2](#_Toc142421494)

[1.4 Đối tượng và phạm vi nghiên cứu 2](#_Toc142421495)

[1.5 Ý nghĩa và ứng dụng 3](#_Toc142421496)

[1.6 Hiện trạng của hệ thống 3](#_Toc142421497)

[CHƯƠNG 2. CƠ SỞ LÝ THUYẾT 4](#_Toc142421498)

[2.1 Flexible 4](#_Toc142421499)

[2.2 Expanded 5](#_Toc142421500)

[2.2.1 Expanded là gì? 5](#_Toc142421501)

[2.2.2 Expanded Constructor 5](#_Toc142421502)

[2.2.3 Child 7](#_Toc142421503)

[2.2.4 Flex 7](#_Toc142421504)

[2.2.5 Flex (ex1) 7](#_Toc142421505)

[2.2.6 Flex (ex2) 8](#_Toc142421506)

[2.3 Call http requests 9](#_Toc142421507)

[2.3.1 Setup 9](#_Toc142421508)

[2.3.2 GET request 9](#_Toc142421509)

[2.3.3 POST request 10](#_Toc142421510)

[2.3.4 PUT request 10](#_Toc142421511)

[2.3.5 PATCH request 11](#_Toc142421512)

[2.3.6 DELETE request 11](#_Toc142421513)

[2.3.7 Authentication 11](#_Toc142421514)

[2.3.8 Basic Authentication 12](#_Toc142421515)

[2.3.9 Bearer (token) Authentication 12](#_Toc142421516)

[2.4 Stack widget 12](#_Toc142421517)

[2.4.1 Stack 12](#_Toc142421518)

[2.4.1.1 Stack Constructor 13](#_Toc142421519)

[2.4.1.2 (ex1) 14](#_Toc142421520)

[2.4.1.3 (ex2) 15](#_Toc142421521)

[2.4.1.4 (ex3) 16](#_Toc142421522)

[2.4.2 Children 17](#_Toc142421523)

[2.4.3 Alignment 17](#_Toc142421524)

[2.4.4 TextDirection 18](#_Toc142421525)

[2.4.5 Fit 18](#_Toc142421526)

[2.4.6 Overflow 18](#_Toc142421527)

[2.4.6.1 overflow (ex1) 19](#_Toc142421528)

[2.4.6.2 overflow (ex2) 20](#_Toc142421529)

[2.4.7 ClipBehavior 21](#_Toc142421530)

[2.5 Future 21](#_Toc142421531)

[2.6 async-socket.io 22](#_Toc142421532)

[2.6.1 Tạo bộ sưu tập API 23](#_Toc142421533)

[2.6.2 Tạo một Server 24](#_Toc142421534)

[2.6.3 Tạo một Client 25](#_Toc142421535)

[2.6.4 Bắt đầu giao tiếp với nhau 26](#_Toc142421536)

[2.7 Await 27](#_Toc142421537)

[2.7.1 Từ khoá “Await” 27](#_Toc142421538)

[2.7.2 Tác dụng 27](#_Toc142421539)

[2.7.3 Cách sử dụng 28](#_Toc142421540)

[2.7.4 Xử lý lỗi khi trong Await 29](#_Toc142421541)

[2.7.5 Hỗ trợ trình duyệt 30](#_Toc142421542)

[CHƯƠNG 3. CÀI ĐẶT THỰC NGHIỆM 31](#_Toc142421543)

[3.1 Cài đặt triển khai 31](#_Toc142421544)

[3.2 Sơ đồ chat realtime websocket 34](#_Toc142421545)

[3.2.1 Giao diện tài khoản người dùng 35](#_Toc142421546)

[3.2.2 Giao diện chọn người muốn hội thoại 35](#_Toc142421547)

[3.2.3 Giao diện camera chụp ảnh và quay video 36](#_Toc142421548)

[3.2.4 Giao diện cuộc hội thoại 37](#_Toc142421549)

[3.2.5 Giao diện thông báo 38](#_Toc142421550)

[3.2.6 Giao diện chọn quốc gia 39](#_Toc142421551)

[3.2.7 Giao diện xác thực người dùng 39](#_Toc142421552)

[CHƯƠNG 4. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN 43](#_Toc142421553)

[4.1 Kết luận 43](#_Toc142421554)

[4.2 Hướng phát triển 43](#_Toc142421555)

[TÀI LIỆU THAM KHẢO 45](#_Toc142421556)

**DANH MỤC CÁC BẢNG, SƠ ĐỒ, HÌNH**

[2. 1 Sơ đồ Expanded 6](#_Toc142472944)

[2. 2 Sơ đồ Expanded Constructor 1 6](#_Toc142472945)

[2. 3 Sơ đồ Expanded Constructor 2 7](#_Toc142472946)

[2. 4 Sơ đồ Flex 8](#_Toc142472947)

[2. 5 Sơ dồ Flex (ex1) 9](#_Toc142472948)

[2. 6 Các phương thức Call http requests 10](#_Toc142472949)

[2. 7 Cài đặt thư viện 10](#_Toc142472950)

[2. 8 Import thư viện 10](#_Toc142472951)

[2. 9 GET request 11](#_Toc142472952)

[2. 10 Post request 11](#_Toc142472953)

[2. 11 PUT request 12](#_Toc142472954)

[2. 12 PATCH request 12](#_Toc142472955)

[2. 13 DELETE request 13](#_Toc142472956)

[2. 14 Basic Authentincation 13](#_Toc142472957)

[2. 15 Bearer (token) Authentication 14](#_Toc142472958)

[2. 16 Mô hình Stack 14](#_Toc142472959)

[2. 17 Sơ đồ Stack Widget 15](#_Toc142472960)

[2. 18 Sơ đồ Stack Construction 1 16](#_Toc142472961)

[2. 19 Sơ đồ Stack Construction 2 17](#_Toc142472962)

[2. 20 Sơ đồ Stack Construction 3 18](#_Toc142472963)

[2. 21 Sơ đồ Alignment 19](#_Toc142472964)

[2. 22 Sơ đồ TextDirection 21](#_Toc142472965)

[2. 23 Sơ đồ overflow(ex1) 22](#_Toc142472966)

[3. 7 Sơ đồ chat realtime trong socket.io 36](#_Toc142472967)

[3. 8 Giao diện chọn tài khoản người dùng 37](#_Toc142472968)

[3. 9 Giao diện chọn người muốn hội thoại 37](#_Toc142472969)

[3. 10 Giao diện camera 38](#_Toc142472970)

[3. 11 Giao diện sau khi chụp ảnh 38](#_Toc142472971)

[3. 12 Giao diện người dùng A 39](#_Toc142472972)

[3. 13 Giao diện người dùng B 39](#_Toc142472973)

[3. 14 Giao diện thông báo 40](#_Toc142472974)

[3. 15 Giao diện chọn quốc gia 41](#_Toc142472975)

[3. 16 Giao diện nhập số điện thoại 41](#_Toc142472976)

[3. 17 Giao diện nhập mã OTP 42](#_Toc142472977)

[3. 18 Giao diện chọn liên hệ 42](#_Toc142472978)

[3. 19 Giao diện tạo nhóm 43](#_Toc142472979)

[3. 20 Giao diện thông tin cuộc gọi 43](#_Toc142472980)

[3. 21 Giao diện hiển thị trạng thái hoạt động 44](#_Toc142472981)

**KÍ HIỆU CÁC CỤM TỪ VIẾT TẮT**

HTTP: Hypertext Transfer Protocol

JSON: JavaScript Object Notation

API: Application Programming Interface

Web: World Wide Web

OTP: One time pasword

1. TỔNG QUAN
   1. 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, Học hỏi và ứng dụng công nghệ mới: Xây dựng một ứng dụng chat sử dụng Flutter và Socket.IO đòi hỏi bạn học hỏi về hai công nghệ này. Flutter là một framework phát triển ứng dụng di động đa nền tảng ngày càng phổ biến, trong khi Socket.IO là một thư viện cho phép giao tiếp thời gian thực. Việc tìm hiểu và áp dụng những kiến thức mới sẽ giúp bạn mở rộng kiến thức kỹ thuật của mình.

Thứ ba, 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 tư, ứng dụng thực tế và thị trường: Ứng dụng chat là một phần quan trọng của cuộc sống hiện đại, được sử dụng rộng rãi cho cả mục đích cá nhân và doanh nghiệp. Việc xây dựng một ứng dụng chat có thể giúp bạn áp dụng kiến thức vào một ứng dụng thực tế và thậm chí có khả năng phát triển thành một sản phẩm thương mại.

Thứ năm, 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.

* 1. 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.
  1. Mục tiêu và nhiệm vụ

Tìm hiểu về Flutter và Socket.io: Nghiên cứu sâu về cấu trúc, tính năng và khả năng của Flutter và Socket.io, để hiểu rõ hơn về việc sử dụng hai công nghệ này để xây dựng ứng dụng chat.

Thiết kế giao diện người dùng hấp dẫn: Xây dựng giao diện người dùng thân thiện, tối ưu hóa trải nghiệm người dùng, đảm bảo tính thẩm mỹ và sự dễ sử dụng của ứng dụng

Xây dựng tính năng chat thời gian thực: Phát triển chức năng chat trực tiếp và mượt mà giữa các người dùng, cho phép gửi tin nhắn và nhận phản hồi trong thời gian thực

Bảo mật và quản lý thông tin người dùng: Đảm bảo tính bảo mật và bảo vệ thông tin cá nhân của người dùng trong quá trình truyền thông và lưu trữ dữ liệu

* 1. Đối tượng và 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.

* 1. Ý nghĩa và ứng dụng

Việc nghiên cứu và phát triển ứng dụng chat bằng Flutter và Rocket.ib sẽ mang lại nhiều ý nghĩa và ứng dụng trong thực tế. Đầu tiên, việc tìm hiểu và ứng dụng Flutter và Rocket.ib trong dự án này sẽ giúp chúng ta hiểu rõ hơn về khả năng và tiềm năng của hai công nghệ này trong việc xây dựng các ứng dụng di động phức tạp

Ứng dụng chat sẽ mang lại lợi ích lớn cho các cộng đồng, tổ chức và doanh nghiệp. Nó giúp nâng cao khả năng giao tiếp và sự kết nối giữa cá nhân và nhóm, cải thiện sự tương tác và hỗ trợ việc trao đổi thông tin một cách thuận tiện và nhanh chóng

* 1. 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 THUYẾT
   1. Flexible

Hãy tưởng tượng, chúng ta muốn làm một giao diện có các widget con bên trong **có thể thay đổi kích thước** theo widget cha (không nên nhầm lẫn với việc size màn hình thay đổi nhé), như vậy thì không thể sử dụng các widget có size cố định, chúng ta hãy nghĩ ngay đến việc sử dụng 3 widget trên :

Ví dụ, chúng ta có 1 Row, bên trong gồm có 4 Container chứa Text:

* Container 1 chiếm 10%
* Container 2 chiếm 20%
* Container 3 chiếm 30%
* Container 4 chiếm 40%

SizedBox(

          height: 100,

          child: Row(

            children: [

              Flexible(

                  flex: 1, child: Container(color: Colors.red.shade100)),

              Flexible(

                  flex: 2,

                  child: Container(

                    color: Colors.red.shade200,

                  )),

              Flexible(

                  flex: 3,

                  child: Container(

                    color: Colors.red.shade300,

                  )),

              Flexible(

                  flex: 4,

                  child: Container(

                    color: Colors.red.shade400,

                  )),

            ],

          ),

        ),

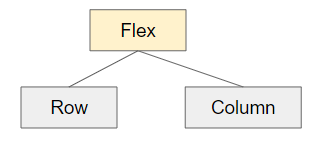
Kích thước của children sẽ được tính như sau:

* Tính tổng hệ số flex của các widget có trong parent (Bao gồm Flexible, Spacer và Expanded): ví dụ ở trên sẽ là 1 + 2 + 3 + 4 = 10
* Đặt kích thước của widget theo tỉ lệ hệ số flex / tổng, Ví dụ Container 1 sẽ chiếm 1/10 tức 10% width của row

Vậy là chúng ta đã có một giao diện động mà không cần sử dụng các kích thước cố định

* 1. Expanded
     1. Expanded là gì?

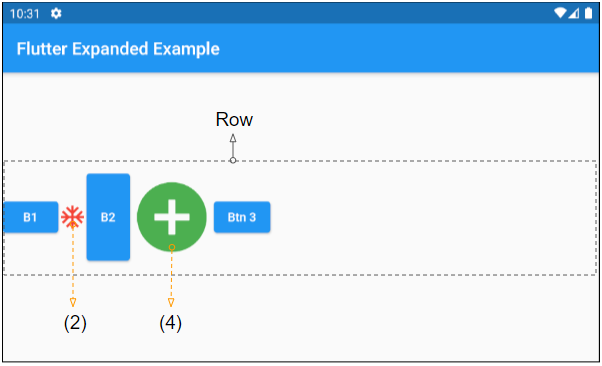
Expanded là một widget giúp mở rộng không gian cho một widget con của Row hoặc Column theo trục chính (main axis). Chú ý, trục chính của Row là trục nằm ngang, và trục chính của Column là trục thẳng đứng.



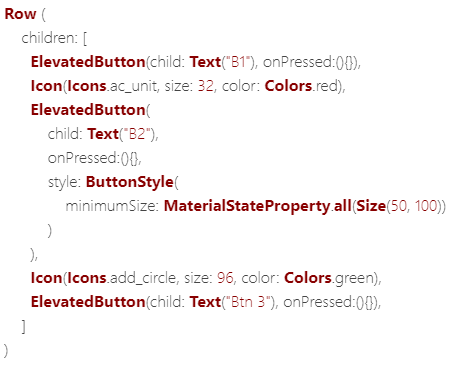
2. 1 Sơ đồ Expanded

* + 1. Expanded Constructor

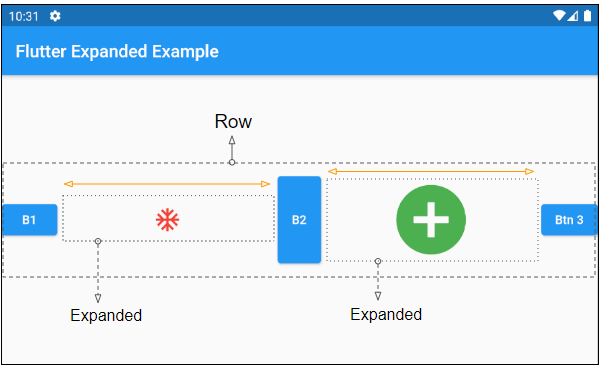
Trước hết, hãy xem một ví dụ như hình minh họa dưới đây. Trong ví dụ này, một đối tượng Row có 5 widget con, câu hỏi là làm thế nào để mở rộng không gian theo phương ngang cho widget con thứ 2 và thứ 4.



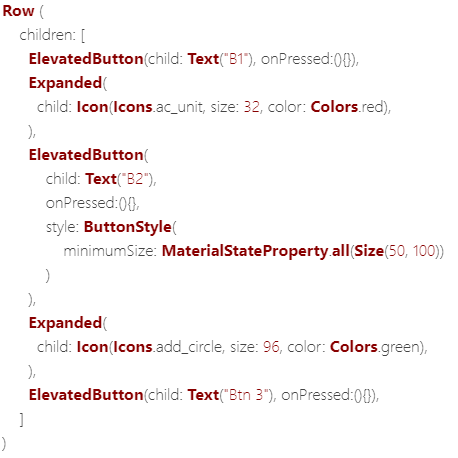
2. 2 Sơ đồ Expanded Constructor 1



Gói một widget con của Row trong một đối tượng Expanded sẽ giúp nó mở rộng không gian theo phương ngang và chiếm phần không gian còn lại của Row



2. 3 Sơ đồ Expanded Constructor 2



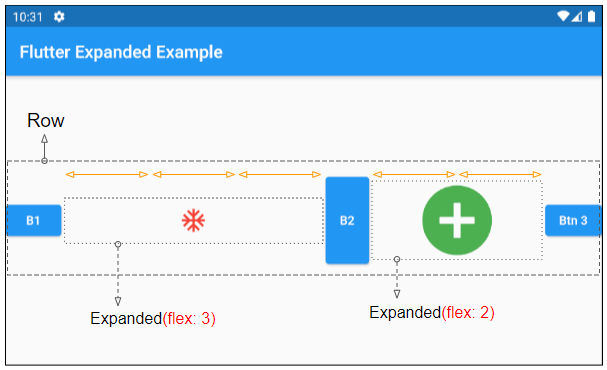
* + 1. Child



* + 1. Flex

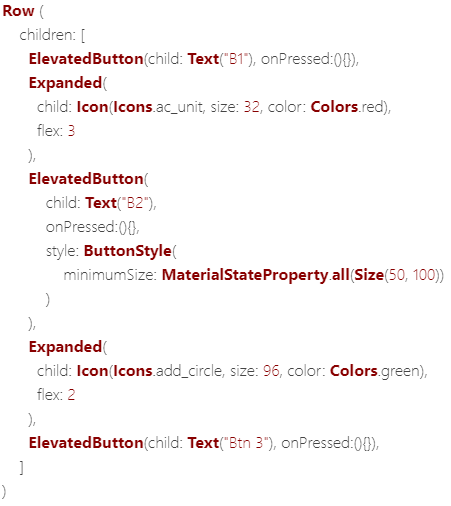
Property flex được coi là trọng lượng của Expanded, nó quyết định bao nhiêu không gian sẽ được phân bổ cho Expanded. Không gian được phân bổ tỷ lệ thuận với giá trị của flex. Giá trị mặc định của flex là 1.

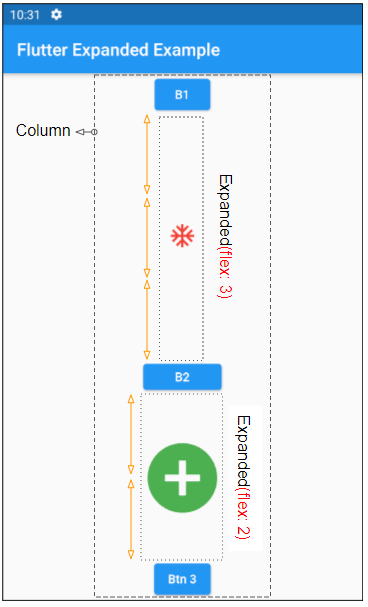
* int flex: 1



2. 4 Sơ đồ Flex

* + 1. Flex (ex1)





2. 5 Sơ dồ Flex (ex1)

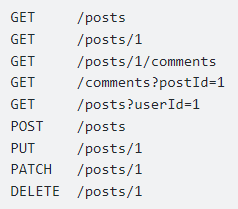
* + 1. Flex (ex2)



* 1. Call http requests

Flutter là một framework sử dụng ngôn ngữ Dart, dùng để lập trình cross-platform cho Android, iOS, desktop application, web app... Hôm nay mình sẽ giới thiệu cho các bạn cách để tạo một request HTTP, sử dụng http package.

Chúng ta sẽ sử dụng JSONPlaceholder làm target cho ví dụ API dưới đây:



2. 6 Các phương thức Call http requests

* + 1. Setup

Thêm http package dependency vào file pubspec.yaml.



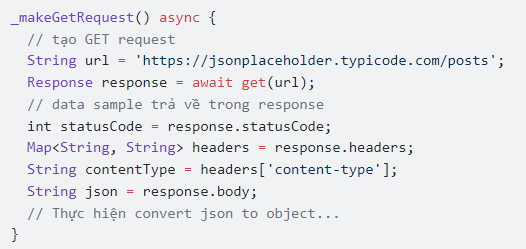
2. 7 Cài đặt thư viện

Trong mỗi file sử dụng package này, import library bằng cách thêm:



2. 8 Import thư viện

* + 1. GET request

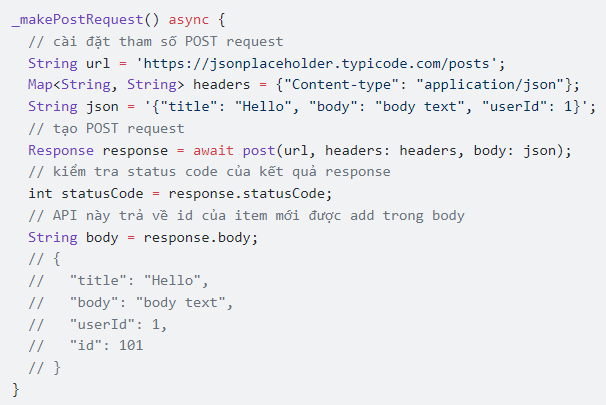


2. 9 GET request

Bây giờ thay thế /posts bằng /post/1 trong url. Sử dụng /posts trả về một array các object JSON trong khi /posts/1 trả về một đối tượng JSON duy nhất, trong đó 1 là ID của post muốn get về. Bạn có thể sử dụng dart:convert để convert JSON raw string to object. Đọc bài viết này để biết thêm chi tiết.

* + 1. POST request

Một request POST dùng để tạo thêm một resource.



2. 10 Post request

* + 1. PUT request

PUT request dùng để thay thế một resource hoặc tạo mới nếu resource đó chưa tồn tại:



2. 11 PUT request

* + 1. PATCH request

PATCH request dùng để sửa đổi một resource có sẵn:



2. 12 PATCH request

Cần lưu ý rằng chuỗi JSON truyền vào chỉ có title, không phải tất cả các phần khác như ví dụ về PUT request.

* + 1. DELETE request

DELETE request dùng để xoá resource:

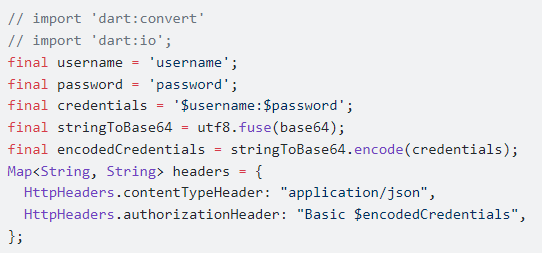


2. 13 DELETE request

* + 1. Authentication

Mặc dù trong ví dụ demo ở trên không yêu cầu authen, nhưng thực tế khi thực hiện request, chúng ta sẽ phải thêm authentication header, đặc biệt khi thực hiện những việc khác ngoài GET request. Có thể add header cho bất kì request nào, mặc dù thông thường thông tin đăng nhập cơ bản (username và password) sẽ được thêm vào POST request ban đầu (để đăng nhập) và sau đó sẽ sử dụng token được trả lại để thêm vào các request tiếp theo.

* + 1. Basic Authentication



2. 14 Basic Authentincation

Trong thiết lập OAuth, bạn có thể đặt client ID và secret trong auth header và username và password trong body. Đây là những gì máy chủ Dart Aqueduct làm, nhưng điều này sẽ phụ thuộc vào thiết lập máy chủ.

Lưu ý rằng tôi đã sử dụng các hằng số HttpHeaders ở đây, yêu cầu dart:io import. Thay vào đó, hãy sử dụng chuỗi "content-type" và "authorization".

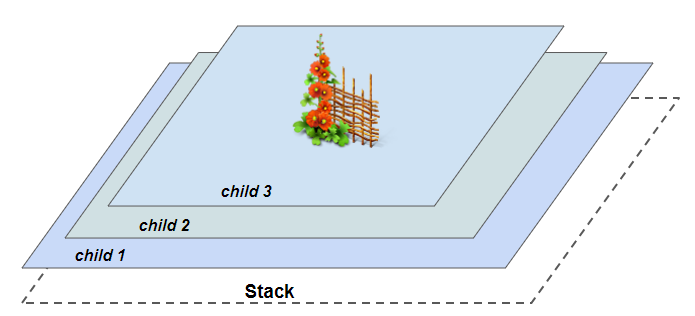
* + 1. Bearer (token) Authentication



2. 15 Bearer (token) Authentication

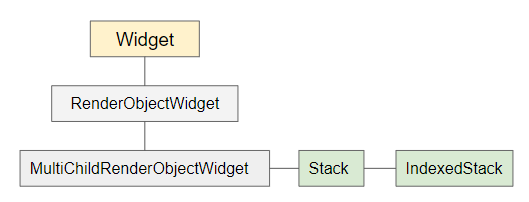
* 1. Stack widget
     1. Stack

Trong Flutter, Stack là một bộ chứa cho phép đặt các widget con của nó chồng lên nhau, widget con đầu tiên sẽ được đặt ở dưới cùng. Stack là một giải pháp để tiết kiệm không gian của ứng dụng. Bạn có thể thay đổi thứ tự của các widget con để tạo ra hiệu ứng hoạt hình đơn giản.



2. 16 Mô hình Stack

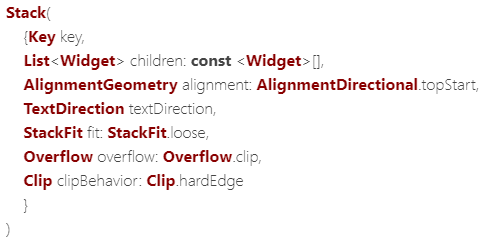
Stack xắp xếp các widget con của nó theo nguyên tắc: widget con đầu tiên sẽ được đặt ở dưới cùng, widget con mới nhất sẽ được đặt ở trên cùng. Khi bạn thay đổi thứ tự của các widget con thì Stack sẽ được vẽ lại. Nếu số lượng và thứ tự các widget con thay đổi liên tục, mỗi widget con cần được cung cấp một giá trị Key cụ thể và duy nhất, điều này giúp Stack quản lý hiệu quả các widget con.



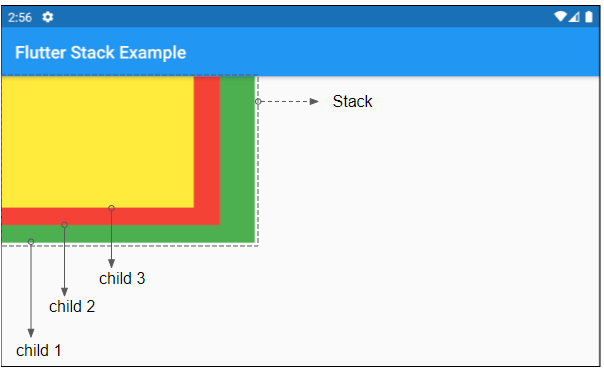
2. 17 Sơ đồ Stack Widget

IndexedStack là một lớp con của Stack. Khác với Stack, tại một thời điểm IndexedStack chỉ hiển thị nhiều nhất một widget con, các widget con khác sẽ bị ẩn. Bạn có thể chỉ định widget con nào sẽ được hiển thị thông qua property index, nếu giá trị của index là null sẽ không có widget con nào được hiển thị.

* + - 1. Stack Constructor

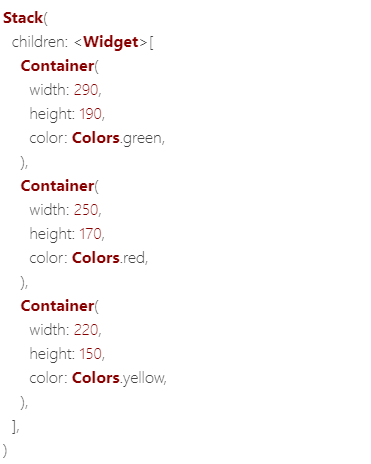


Dưới đây là một ví dụ đơn giản, một Stack với 3 widget con, theo mặc định chúng được định vị ở góc trên bên phải của Stack, bạn có thể nhìn thấy chúng chồng chéo lên nhau:



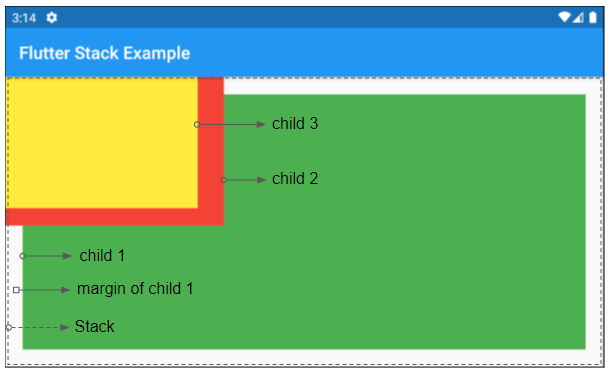
2. 18 Sơ đồ Stack Construction 1

* + - 1. (ex1)



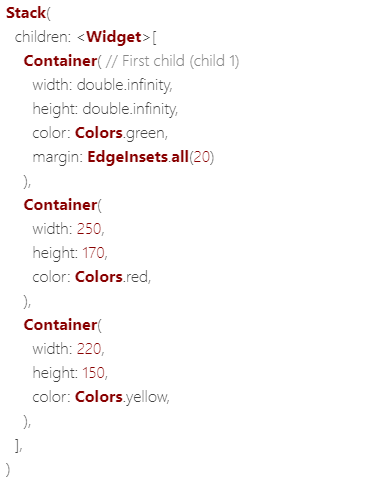
Về cơ bản, Kích thước của Stack là nhỏ nhất có thể, và cố gắng lớn hơn tất cả các widget con của nó (Ngoại trừ các widget con là Positioned hoặc Transform, xem thêm property overflow).

Hãy xem một ví dụ: Một Stack với một widget con có kích thước tối đa



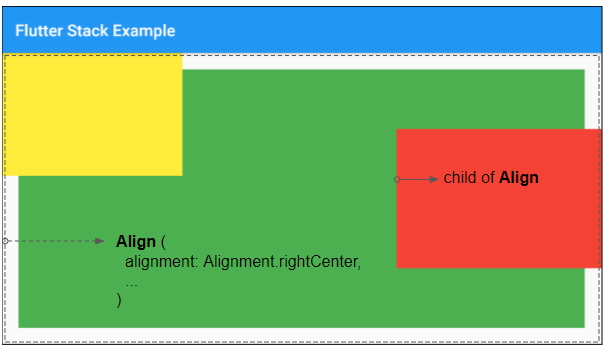
2. 19 Sơ đồ Stack Construction 2

* + - 1. (ex2)



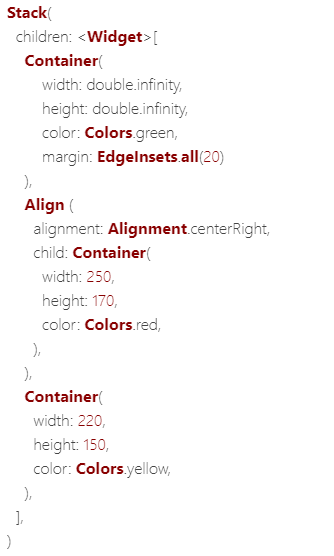
Nếu bạn muốn căn chỉnh vị trí của một widget con (của Stack) hãy đặt nó trong một Align

Chú ý: Khi widthFactor và heightFactor không được chỉ định, nhưng child được chỉ định thì Align sẽ có kích thước lớn nhất có thể.



2. 20 Sơ đồ Stack Construction 3

* + - 1. (ex3)



* + 1. Children

children - là một danh sách các widget con của Stack.

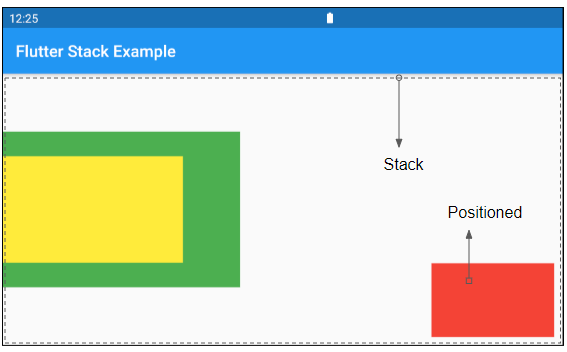


* + 1. Alignment

Property alignment được sử dụng để căn chỉnh các widget con không phải Positioned. Giá trị mặc định của nó là AlignmentDirectional.topStart.



Ví dụ dưới đây cho thấy property alignment của Stack không có tác dụng đối với các widget con là Positioned.



2. 21 Sơ đồ Alignment



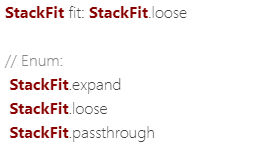
* + 1. TextDirection

Property textDirection được sử dụng để sét đặt hướng của văn bản, giá trị của nó sẽ ảnh hưởng tới hành vi của property alignment.



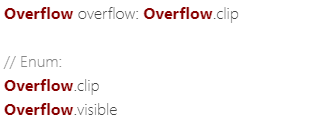
* + 1. Fit

Property fit chỉ ra cách "Làm thế nào để định kích thước cho các widget con không phải Positioned của IndexedStack". Giá trị mặc định của nó là StackFit.loose

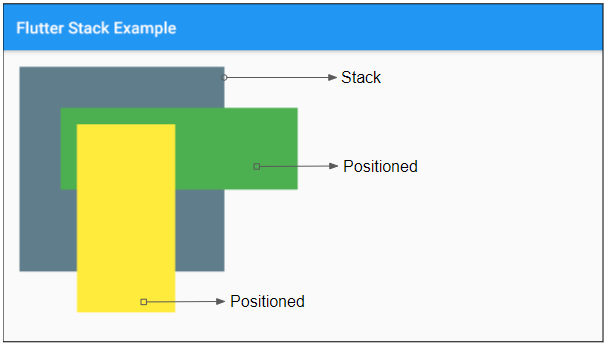


* + 1. Overflow

Property overflow được sử dụng để chỉ định cách Stack sẽ đối xử với các widget con tràn ra (overflow) khỏi nó. Cách đối xử có thể là xén (clip) phần tràn ra ngoài, hoặc cho phép phần tràn ra ngoài hiển thị. Giá trị mặc định của overflow là Overflow.clip.

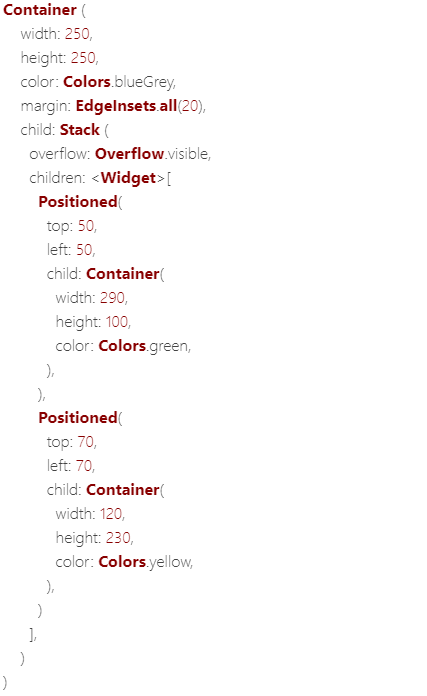


Các kết quả kiểm tra cho thấy overflow:Overflow.visible chỉ làm việc với vài loại widget con (Chẳng hạn Positioned hoặc Transform).

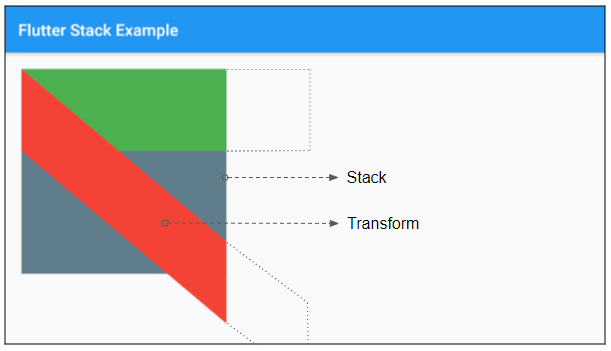


2. 22 Sơ đồ TextDirection

* + - 1. overflow (ex1)



Ví dụ: Một Transform xiên (skew) dọc theo trục Y, nó có thể bị tràn ra (overflow) khỏi Stack, phần tràn theo trục Y sẽ được hiển thị, phần tràn theo trục X sẽ bị xén (clip).



2. 23 Sơ đồ overflow(ex1)

* + - 1. overflow (ex2)



* + 1. ClipBehavior



* 1. Future

Một future("f" viết thường) là một thể hiện của class Future . Một future đại diện cho một hoạt động bất đồng bộ, và có thể có hai trạng thái: Uncompleted hoặc Completed.

Uncompleted: khi bạn gọi một hàm không đồng bộ, nó trả về một uncompleted future. Future này đang chờ cho các hoạt động không đồng bộ của hàm kết thúc hoặc trả về một error.

Completed: nếu một hành động không đồng bộ thành công, future sẽ hoàn thành với một giá trị. Nếu không nó hoàn thành với một error.

* Hoàn thành với một giá trị: một future của kiểu Future<T> hoàn thành với một giá trị thuộc kiểu T. Ví dụ, một future với kiểu Future<String> thì nó sẽ trả về một giá trị string. Nếu một future không cung cấp một kiểu giá trị nào thì kiểu của future là Future<void>.
* Hoàn thành với một error: nếu một hoạt động bất đồng bộ được thực hiện bởi một hàm thất bại vì bất kỳ lý do gì thì future hoàn thành với một error.

Ví dụ giới thiệu về future, bạn hãy dự đoán kết quả nhé, nó sẽ in ra dòng nào trước: "Large Latte" hay là "Fetching user order..."?

Future<void> getUserOrder() {

  // Imagine that this function is fetching user

  //info from another service or database

  return Future.delayed(Duration(seconds: 3), () => print('Large Latte'));

}

main() {

  getUserOrder();

  print('Fetching user order...');

}

Kết quả là: nó sẽ in ra "Fetching user order ... " sau 3 giây nó in ra tiếp "Large Latte"

Performing hot restart... Syncing files to device Android SDK built for x86... I/flutter ( 5845): Fetching user order... Restarted application in 974ms. I/flutter ( 5845): Large Latte

**Tóm lược lại một chút:**

* Một thể hiện Future<T> cung cấp một giá trị kiểu T. Nếu một future không cung cấp một giá trị hữu dụng thì kiểu của future là Future<void>.
* Một future có thể có một trong hai trạng thái: uncompleted hoặc completed.
* Khi một hoạt động của future kết thúc, future hoàn thành với một giá trị hoặc một error.
* future là một thể hiện của Dart Future class.
  1. async-socket.io

**async-socket.io** là một framework và phương pháp giúp sử dụng socket.io mà không cần viết bất kỳ mã socket nào. Nó ra đời từ nhu cầu viết nhiều mã giao tiếp giữa clients và server, khiến code trở nên dài dòng và phức tạp nếu phải sử dụng các hàm **socket.on** và **socket.emit()** để xử lý giao tiếp. Điều này khiến mã code dễ trở nên rối và khó bảo trì.

Thay vào đó, async-socket.io cho phép bạn thể hiện các chức năng mà clients của bạn hỗ trợ và các chức năng mà server của bạn hỗ trợ, trong một đối tượng API có namespace như sau:

const API = {

// Namespace quản trị, dành cho các hoạt động quản trị.

admin: {

client: ['register', 'getStateDigest'],

server: []

},

// Namespace người dùng, dành cho các hoạt động liên quan đến người dùng

user: {

client: ['userJoined', 'userLeft'], server: ['setName', 'getUserList']

};

Sau đó, bạn chạy API qua framework async-socket.io để tự động tạo ra một tập hợp các đối tượng proxy, đồng thời xử lý tất cả các mã socket.io và ẩn việc sử dụng sockets hoàn toàn, cho phép viết code như nếu clients và server có tham chiếu trực tiếp đến nhau.

Bạn sẽ thấy từ khóa **async** quan trọng, bởi vì để cho phép xử lý socket tự động cũng như định tuyến dữ liệu tự động, tất cả các hàm mà bạn hứa sẽ tồn tại trong API phải được khai báo là **async**, giúp framework xử lý giao tiếp socket dưới dạng promises, bao gồm cả đăng ký và hủy đăng ký các sự kiện phản hồi.

Điều quan trọng là khi sử dụng framework này, bạn không cần suy nghĩ về việc sử dụng sockets bằng bất kỳ cách nào ngoài việc chắc chắn rằng bạn truyền giá trị **io** và **socket** của socket.io vào các hàm đúng.

* + 1. Tạo bộ sưu tập API

Như đã đề cập ở trên, một bộ sưu tập API được tạo ra bằng cách xác định một đối tượng API với namespace, sau đó chạy qua trình biến đổi **generateClientServer**:

const generateClientServer = require( "async-socket.io');

const API = {

user: (

client: [ 'register', ...],

server: ['setName', ... ]

};

const ClientServer = generateClientServer(API);

Trong ví dụ này, tạo một bộ sưu tập API cho namespace **"user"**. API này bao gồm các hàm mà client hỗ trợ (client) và các hàm mà server hỗ trợ (server).

Sau khi chạy API qua hàm **generateClientServer**, sẽ có một đối tượng **ClientServer** chứa các đối tượng proxy đã được tự động tạo ra bởi **async-socket.io**. Các đối tượng proxy này sẽ xử lý tất cả mã socket.io và ẩn việc sử dụng sockets, cho phép code được viết như nếu clients và server có tham chiếu trực tiếp đến nhau.

* + 1. Tạo một Server

Sau khi có mã trên, bạn có thể tạo một lớp Server để xử lý thực sự các call API, bao gồm triển khai cho hàm bắt buộc **addClient(client)**, và sau đó tạo một máy chủ websocket với một lời gọi đơn:

class ServerClass {

constructor() {

this.clients = [];

}

addClient(client) {

this.clients.push(client);

let clientId = this.clients.length;

client.admin.register(clientId);

}

async setName(from, name) {

let client = this.clients.find(c => c == from);

client.name = name;

}

}

Trong ví dụ trên, tạo một lớp **ServerClass** để xử lý các call API thực tế. Lớp này có một mảng clients để lưu trữ danh sách các client đã kết nối. Hàm **addClient(client)** được triển khai để thêm một client vào danh sách và gán một clientId cho client đó bằng cách gọi hàm **client.admin.register(clientId).**

Hàm **setName(from, name)** được triển khai để đặt tên cho client, trong đó from là tham chiếu đến client đã gọi API. Sau đó, server tìm client tương ứng trong danh sách và gán tên mới cho nó.

Tiếp theo, sử dụng lớp **ServerClass** đã tạo để triển khai máy chủ:

const server = ClientServer.createServer(ServerClass); server.listen(0, () => console.log('started server );

Trong đoạn mã trên, sử **dụng ClientServer.createServer(ServerClass)** để tạo một máy chủ sử dụng ServerClass đã triển khai. Sau đó, lắng nghe trên cổng 0 (tức là sẽ chọn một cổng ngẫu nhiên), và khi máy chủ bắt đầu lắng nghe, in ra thông báo về cổng được sử dụng.

Lưu ý rằng tất cả các hàm xử lý API trong lớp ServerClass đều được truyền một tham chiếu đến client đã gọi API dưới dạng tham số **from**. Điều này áp dụng đối với tất cả các hàm xử lý call API trong lớp ServerClass, và tham số này sẽ luôn được truyền làm tham số đầu tiên cho bất kỳ hàm xử lý call API nào.

Nếu máy khách gọi **server.doThing(data)**, máy chủ phải có chức năng xử lý bằng chữ ký **async doThing(from, data) { ... }**

* + 1. Tạo một Client

Việc tạo một client tương tự như việc tạo một server:

const server = ClientServer.createServer(ServerClass); server.listen(0, () => console.log('started server );class ClientClass {

constructor() {

this.id = -1;

async register(clientId) {

this.id = clientId;

let name = this.name = generateRandomName();

this.server.user.setName(name);

}

}

Trong ví dụ trên, tạo một lớp **ClientClass** để triển khai các call API của client. Lớp này có một thuộc tính id để lưu trữ clientId của client và một phương thức **register(clientId)** để đăng ký client với server. Khi client đăng ký thành công, nó sẽ gán clientId cho thuộc tính id của chính nó, sau đó sử dụng hàm **generateRandomName()** để tạo một tên ngẫu nhiên cho chính nó. Sau đó, nó gọi hàm **this.server.user.setName(name)** để thông báo tên của nó cho server thông qua API **setName**.

Sau khi triển khai lớp **ClientClass**, sẽ tạo một (hoặc nhiều) Client khi máy chủ đã sẵn sàng:

server.listen(0, () => { const serverURL = http://\*:$(server.address().port}'; ClientServer.createClient(serverURL, ClientClass); });

Trong đoạn mã trên, đang lắng nghe máy chủ trên cổng 0 (tức là sẽ chọn một cổng ngẫu nhiên), và khi máy chủ bắt đầu lắng nghe, tạo một client bằng cách sử dụng **ClientServer.createClient(serverURL, ClientClass)**. Điều này sẽ tạo một client mới, kết nối với máy chủ ở địa chỉ **serverURL** và sử dụng lớp **ClientClass** mà đã triển khai trước đó để xử lý các call API từ server.

Lưu ý rằng các hàm xử lý call API cho client không nhận tham số **from**, vì client đã kết nối đến một server duy nhất. Nguồn call luôn được biết đến và proxy của server luôn có thể được tham chiếu bằng cách sử dụng **this.server** bên trong bất kỳ hàm xử lý call API nào.

* + 1. Bắt đầu giao tiếp với nhau

Hãy xem thư mục "demo" để xem một ví dụ về cách thiết lập một hệ thống đơn giản với server và ba client. Trong ví dụ này, có code để khởi động một máy chủ và ba client.

Khi client kết nối thành công, server sẽ thông báo cho mỗi client về **id** của họ, thêm họ vào danh sách các người dùng đã biết, và mỗi client sẽ tự tạo một tên ngẫu nhiên cho chính họ khi đăng ký, sau đó thông báo tên này cho server. Sau đó, mỗi client sẽ yêu cầu server cung cấp danh sách người dùng mà server đang giữ, và tự động nhận thông báo khi có các hành động tham gia/rời khỏi từng người dùng riêng lẻ xảy ra.

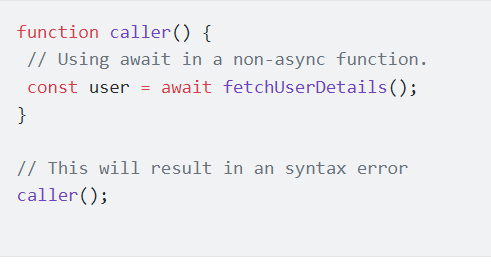
Bạn có thể chạy demo này bằng cách sử dụng lệnh **"npm test"** trong thư mục **async-socket.io.**

Bạn cũng có thể chạy kiểm thử này bằng cách sử dụng các quy trình độc lập cho client và server, bằng cách sử dụng lệnh **"npm test:distributed"**, tuy nhiên điều này sẽ tạo ra các tiến trình độc lập thực sự và chủ yếu được sử dụng để chỉ ra rằng "mọi thứ hoạt động" chứ không cung cấp cách dễ dàng để xem xét chi tiết cụ thể.

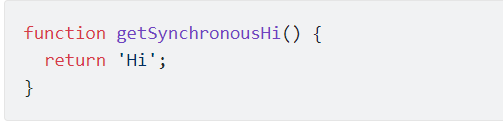
Tóm lại, qua demo này, bạn sẽ thấy cách tạo một hệ thống giao tiếp giữa server và các client thông qua **async-socket.io** framework một cách dễ dàng và hiệu quả, giúp bạn tập trung vào việc triển khai chức năng hơn là quản lý mã socket.io phức tạp.

* 1. Await
     1. Từ khoá “Await”
* Trong ngữ cảnh của JavaScript và Python, "await" là một từ khóa được sử dụng trong các hàm bất đồng bộ (asynchronous functions) để thực hiện việc chờ đợi kết quả của một tác vụ bất đồng bộ trước khi tiếp tục thực hiện các tác vụ khác. Điều này giúp tránh chặn (blocking) các hoạt động trong khi chờ đợi một hàm bất đồng bộ hoàn thành.
* Khi từ khóa "await" được sử dụng, các tác vụ trong hàm sẽ dừng lại tại đó cho đến khi hàm bất đồng bộ trả về kết quả. Trong quá trình chờ đợi, luồng của chương trình không bị chặn và có thể thực hiện các tác vụ khác mà không phụ thuộc vào kết quả của hàm bất đồng bộ.
  + 1. Tác dụng
* Chờ đợi kết quả từ hàm bất đồng bộ: Khi sử dụng "await," chương trình dừng thực thi tại đó và chờ đợi cho kết quả của một hàm bất đồng bộ trước khi tiếp tục thực hiện các dòng lệnh tiếp theo. Điều này giúp đảm bảo rằng kết quả của hàm bất đồng bộ đã sẵn sàng và có sẵn để sử dụng trước khi thực hiện các tác vụ khác.
* Không chặn luồng thực thi: Khi sử dụng "await," các hoạt động khác trong chương trình có thể tiếp tục thực thi mà không bị chặn (blocking) bởi việc chờ đợi kết quả từ hàm bất đồng bộ. Điều này giúp tránh tình trạng đứng im và tăng cường hiệu suất ứng dụng khi làm việc với nhiều hoạt động đồng thời.
* Xử lý mã lỗi dễ dàng: Khi sử dụng "await," bạn có thể sử dụng cấu trúc try-catch để xử lý các lỗi xảy ra trong hàm bất đồng bộ một cách dễ dàng và chính xác hơn.
* Tiết kiệm tài nguyên: Thay vì phải sử dụng các phương pháp khác như callback hay promise chaining, "await" giúp viết mã bất đồng bộ dễ đọc hơn và giảm thiểu việc sử dụng tài nguyên do gom nhóm các hoạt động bất đồng bộ vào một hàm chứa "await".
* Lưu ý rằng "await" chỉ có thể sử dụng trong các hàm "async" và phải được hỗ trợ bởi ngôn ngữ lập trình và các thư viện hỗ trợ bất đồng bộ. Trong JavaScript, "await" được sử dụng trong các hàm "async" khi làm việc với Promise. Trong Python, "await" được sử dụng trong các hàm "async" khi làm việc với asyncio hay các thư viện hỗ trợ bất đồng bộ khác.
  + 1. Cách sử dụng

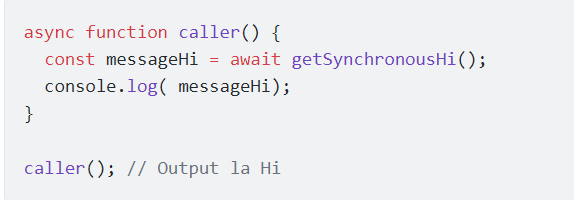
Không thể sử dụng await trong một function thông thường, non-async function. JavaScript engine sẽ tạo ra một syntax error nếu chúng ta cố tình làm như vậy.



Function mà chúng ta sử dụng sau await có thể là một async hoặc không. Chúng ta không có quy tắc bắt buộc nó phải là một function bất đồng bộ nhé. Ví dụ :

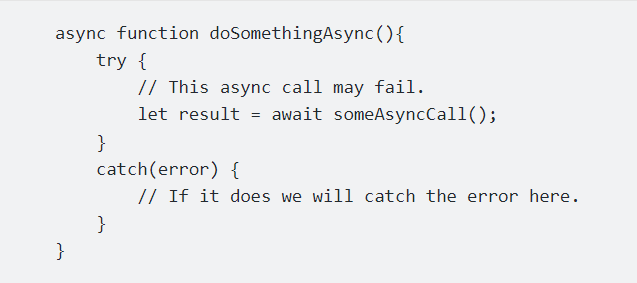


Chúng ta vẫn có thể sử dụng await trong khi gọi function trên :



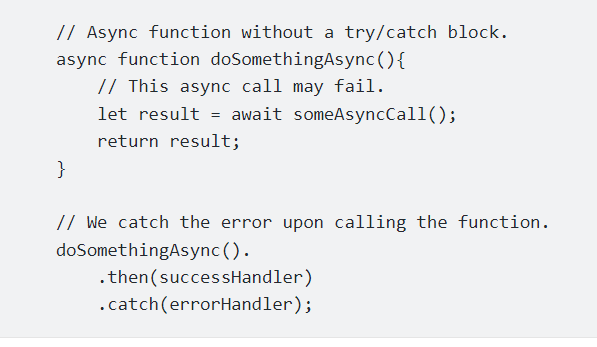
* + 1. Xử lý lỗi khi trong Await

Một điều tuyệt vời khác về Async / Await là nó cho phép chúng ta bắt các lỗi không mong đợi bằng cách sử dụng try / catch. Chúng ta chỉ cần để các await call của chúng ta vào trong khối try/catch như sau:



Mệnh đề catch sẽ xử lý các lỗi gây ra bởi các hàm bất đồng bộ hoặc bất kỳ lỗi nào chúng ta có thể đã viết bên trong khối try.

Trong một vài tình huống, chúng ta cũng có thể bắt các lỗi khi đang thực hiện function async. Vì tất cả các hàm async đều trả về Promises, chúng ta chỉ cần gọi thêm hàm .catch() khi gọi chúng.



Dựa vào các tình huống cụ thể, chúng ta sẽ sử dụng try/catch hoặc .catch() để bắt và xử lý lỗi. Tuy nhiên, chúng ta không nên sử dụng cả 2 cùng một lúc vì nó có thể dẫn đến các vấn đề không mong muốn.

* + 1. Hỗ trợ trình duyệt

Async / Await có thể sử dụng trong hầu hết các trình duyệt chính, ngoại trừ IE11 - tất cả các trình duyệt sẽ nhận ra mã async / await của bạn mà không cần các thư viện bên ngoài.

Nếu các bạn dùng Node cũng có sử dụng async từ Node 8 trở lên. Nó sẽ trở thành LTS vào cuối năm nay.

Nếu cái này không đáp ứng được nhu cầu của bạn, cũng có một số trình chuyển đổi JS như Babel và TypeScript và thư viện Node.js async/await cung cấp các phiên bản nền tảng riêng của tính năng này.

1. CÀI ĐẶT THỰC NGHIỆM
   1. 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:

dependencies:

flutter:

sdk: flutter # Phiên bản mới nhất của thư viện

SocketIO socket\_io\_client: ^x.X.x # Phiên bản mới nhất http: ^x.X.X # Phiên bản mới nhất của thư viện

HTTP websocket: ^x.x.x # Phiên bản mới nhất của thư viện WebSocket

- 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:

import 'package: socket\_io\_client/socket\_io\_client.dart' as 10;

10.Socket socket = 10.io('http://your\_server\_url'); socket.on('connect', (\_) { print('Connected');

});

socket.on('chat\_message', (data) { print('Received message: $data');

});

socket.onDisconnect((\_\_) => print('Disconnected'));

socket.connect();

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:

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

void fetchData() async { var url = Uri.parse('http://your\_api\_url'); var response = await http.get(url);

if (response.statusCode == 200) {

// Xử lý dữ liệu nhận được từ server var data = response.body; print('Received data: $data');

} else {

print('Request failed with status: ${response.statusCode}');

}

}

import 'package:web\_socket\_channel/io.dart';

import 'package:web\_socket\_channel/web\_socket\_channel.dart';

import 'package:web\_socket\_channel/status.dart' as status;

WebSocketChannel channel = IOwebSocketChannel.connect('ws://your\_se

rver\_url');

channel.stream.listen((message) {

print('Received message: $message');

});

channel.sink.add('Hello, server!');

channel.sink.close(status.normalClosure);

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

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

void fetchData() async { var url = Uri.parse('http://your\_api\_url'); var response = await http.get(url);

if (response.statusCode == 200) {

// Xử lý dữ liệu nhận được từ server var data = response.body; print('Received data: $data');

} else {

print('Request failed with status: ${response.statusCode}');

}

}

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:

void handleMessage (dynamic data) {

// xử lý dữ liệu nhận được từ server (ví dụ: tin nhắn mới) print('Received message: $data');

// Cập nhật giao diện người dùng với tin nhắn mới }

void sendMessage(String message) {

// Gửi tin nhắn đến server thông qua webSocket hoặc SocketIO print('Sending message: $message');

// Cập nhật giao diện người dùng với tin nhắn đã gửi

}

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.

void dispose() {

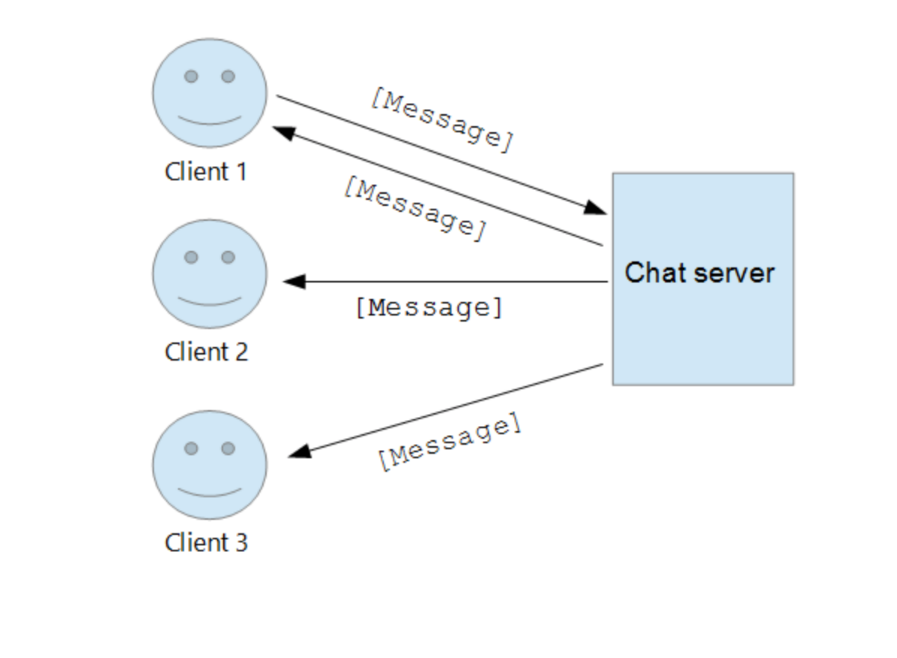
// Đóng kết nối WebSocket hoặc SocketIO socket.disconnect();

// Giải phóng tài nguyên socket.destroy();

}

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. Sơ đồ chat realtime websocket

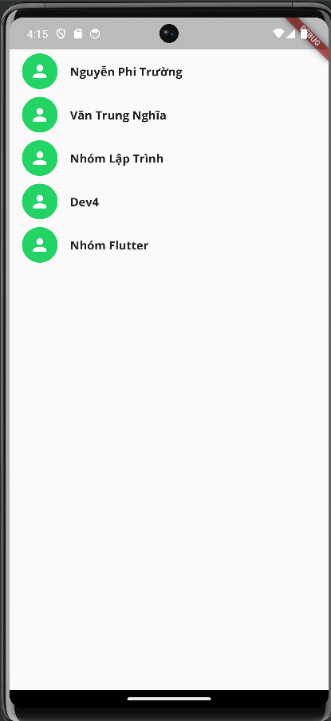


3. 7 Sơ đồ chat realtime trong socket.io

Trong Socket.IO, "gửi event" đề cập đến việc trao đổi dữ liệu và thông điệp giữa các máy tính hoặc thiết bị thông qua kết nối websocket. Các "event" là các thông điệp hoặc sự kiện có thể được gửi từ máy chủ đến máy khách hoặc ngược lại để thực hiện giao tiếp thời gian thực.

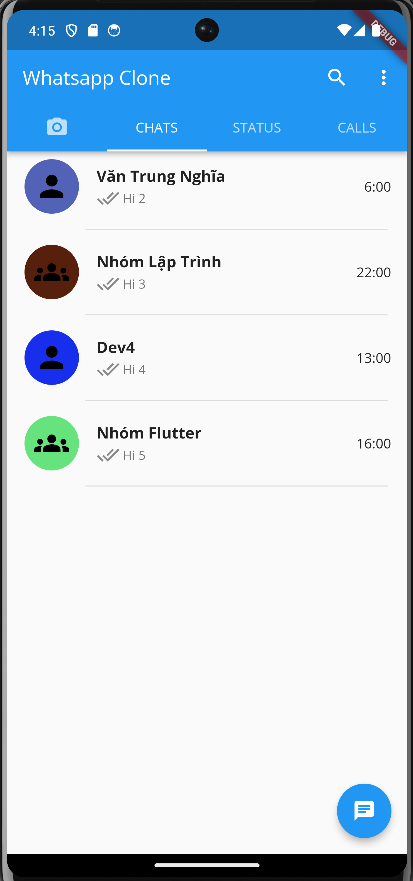
Khi chúng ta xây dựng ứng dụng sử dụng Socket.IO, việc gửi event là một phần quan trọng của quá trình giao tiếp.

* 1. Thiết kế giao diện và chức năng của app
     1. Giao diện tài khoản người dùng



3. 8 Giao diện chọn tài khoản người dùng

* Truy cập vào ứng dụng user chọn tên người dùng vào tài khoản cá nhân của mình tiến hành chat với thư viện socket.io
  + 1. Giao diện chọn người muốn hội thoại



3. 9 Giao diện chọn người muốn hội thoại

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.

* + 1. Giao diện camera chụp ảnh và quay video



3. 10 Giao diện camera

* Người dùng có thể bật đèn flash, quay camera trước hoặc sau hoặc nhấn nút giữa để chụp hay quay video.



3. 11 Giao diện sau khi chụp ảnh

* + 1. Giao diện cuộc hội thoại



3. 12 Giao diện người dùng A

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.



3. 13 Giao diện người dùng B

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.

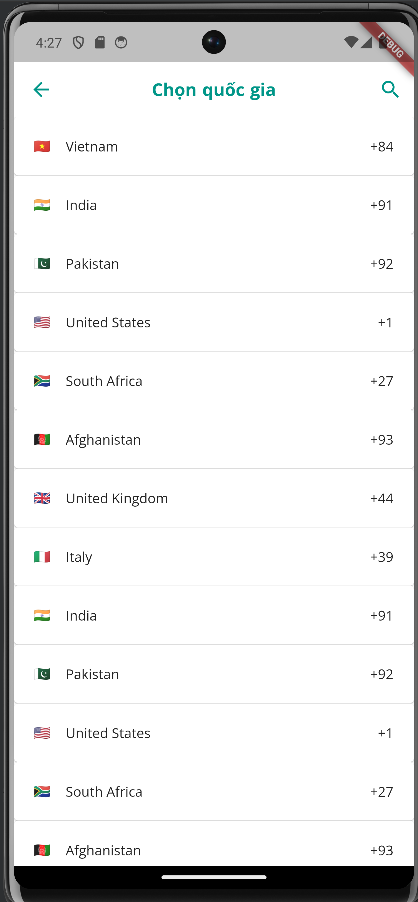
Giao diện trò chuyện thường được thiết kế để đơn giản, trực quan và dễ sử dụng. Nó tập trung vào việc cung cấp các công cụ và chức năng cần thiết để người dùng có thể dễ dàng trò chuyện và tương tác với nhau trong một môi trường trực tuyến.

* + 1. Giao diện thông báo



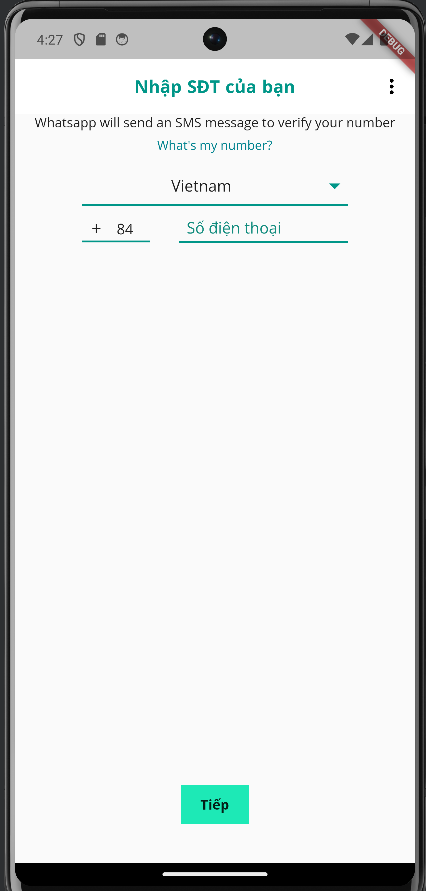
3. 14 Giao diện thông báo

* + 1. Giao diện chọn quốc gia

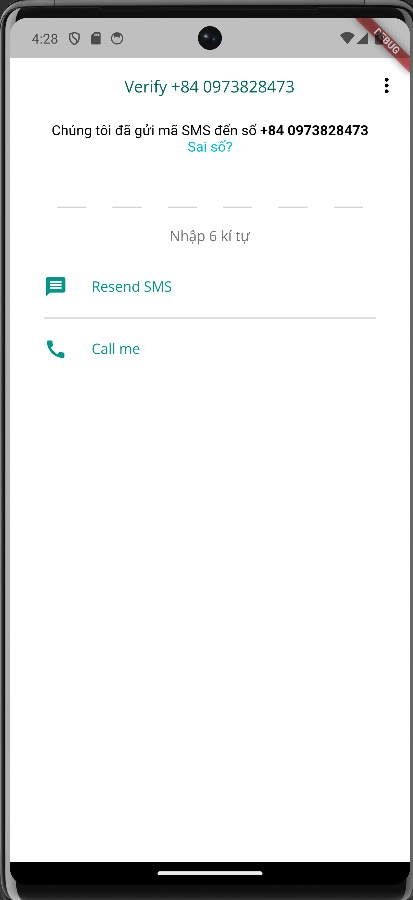


3. 15 Giao diện chọn quốc gia

* + 1. Giao diện xác thực người dùng

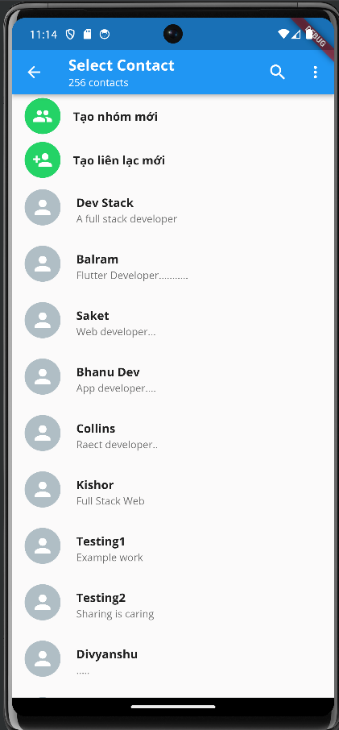


3. 16 Giao diện nhập số điện thoại

**

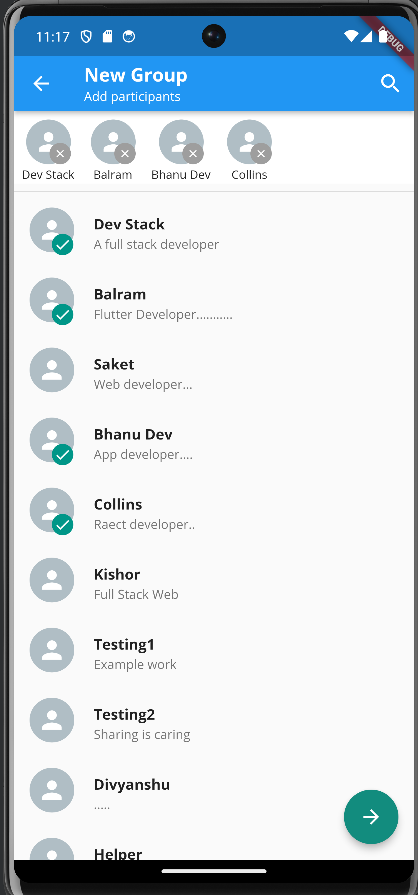
3. 17 Giao diện nhập mã OTP

* Người dùng có thể chọn 2 phương thức xác thực gửi tin nhắn hoặc gọi sau khi đã lấy được mã OTP người dùng nhập vào ứng dụng để đăng nhập vào hệ thống
  + 1. Giao diện chọn liên hệ



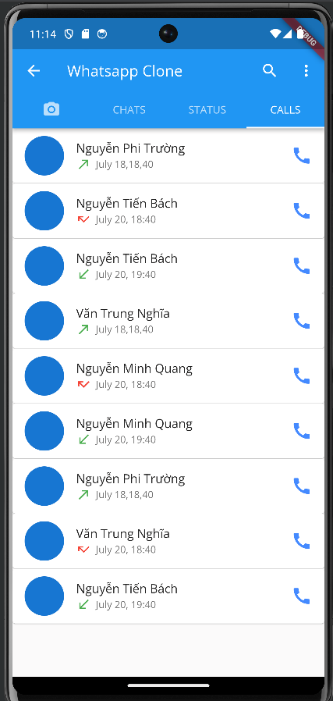
3. 18 Giao diện chọn liên hệ

* Ứng dụng thể hiện danh sách các người dùng chúng ta có thể quen biết sau khi chọn được người dùng chúng ta ấn vào để tiến hành cuộc hội thoại.
  + 1. Giao diện tạo nhóm



3. 19 Giao diện tạo nhóm

* Chúng ta chọn những tài khoản thêm nó vào nhóm để tiến hành cuộc hội thoại chung
  + 1. Giao diện thông tin cuộc gọi



3. 20 Giao diện thông tin cuộc gọi

* Thể hiện danh sách các thông tin người dùng như ngày giờ, trạng thái cuộc gọi.
  + 1. Giao diện hiển thị trạng thái hoạt động



3. 21 Giao diện hiển thị trạng thái hoạt động

1. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN
   1. 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.

***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.

* 1. 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.

* 1. Các câu hỏi

*Các Nhóm Hỏi*

|  |  |
| --- | --- |
| Nhóm 1 | Làm thế nào để xử lý dữ liệu trả về từ HTTP request? |
| Nhóm 2 | Sự khác biệt giữa gửi và nhận dữ liệu thông qua Socket.IO trong Flutter là gì? |
| Nhóm 3 | Làm thế nào để quản lý trạng thái chat khi nhận và hiển thị tin nhắn mới trong thời gian thực |
| Nhóm 4 | Ưu điểm của Socket.io khi làm ứng dụng chat sẽ tốt hơn ở điểm nào so với khi sử dụng BloC Architecture? |

*Câu Hỏi Cho Các Nhóm*

|  |  |
| --- | --- |
| Nhóm 6 | Làm thế nào để tích hợp Socket.io vào ứng dụng chat sử dụng Flutter? |
| Nhóm 7 | Sự khác nhau giữa hàm đồng bộ và hàm bất đồng bộ? |
| Nhóm 8 | Cách đảm bảo tính bảo mật và xác thực trong ứng dụng chat sử dụng Socket.io và Flutter? |
| Nhóm 9 | Cách tối ưu hóa hiệu suất của ứng dụng chat khi sử dụng Socket.io và Flutter? |

TÀI LIỆU THAM KHẢO

[1] Hien, D. T. (2019). *Flutter - Lập trình bất đồng bộ với Future, async, await.* Retrieved from https://viblo.asia/p/flutter-lap-trinh-bat-dong-bo-voi-future-async-await-oOVlYpLaZ8W

[2] openplanning. (n.d.). *Hướng dẫn và ví dụ Flutter Expanded*. Retrieved from https://openplanning.net/13117/flutter-expanded

[3] openplanning. (n.d.). *Hướng dẫn và ví dụ Flutter Stack*. Retrieved from https://openplanning.net/13107/flutter-stack

[4] SonNguyen. (2020). *Tạo HTTP request trong Flutter*. Retrieved from https://viblo.asia/p/tao-http-request-trong-flutter-07LKXmJeZV4

[5] Technologies, G. S. (n.d.). Retrieved from [FLUTTER] Một số widget hữu ích trong Flutter (Phần 1): https://magz.techover.io/2022/04/04/flutter-mot-so-widget-huu-ich-trong-flutter-phan-1/