

BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC NHA TRANG
KHOA CÔNG NGHỆ THÔNG TIN



BÀI TẬP NHÓM
ỨNG DỤNG NHẮN TIN VÀ GỌI ĐIỆN TRỰC TUYẾN

Học phần: Lập trình hướng đối tượng
Lớp học phần: 64.CNTT-3
Giảng viên hướng dẫn: ThS. Phạm Thị Kim Ngoan
Nhóm thực hiện: Nhóm 1

Danh sách thành viên

STT	Họ và tên	MSSV
1	Nguyễn Quang Vinh	64132989
2	Trần Thanh Trí	64132675
3	Lê Ngọc Minh Châu	64130175
4	Phạm Hồ Như Thủy	64132456
5	Nguyễn Trà My	64131348

MỤC LỤC

DANH MỤC HÌNH ẢNH.....	i
DANH MỤC TỪ KHÓA – VIẾT TẮT.....	ii
LỜI MỞ ĐẦU.....	iii
Chương 1. TỔNG QUAN VỀ ĐỀ TÀI	1
1.1. Giới thiệu về ứng dụng nhắn tin và gọi điện trực tuyến	1
1.2. Yêu cầu thực hiện.....	1
Chương 2: THIẾT KẾ VÀ LỰA CHỌN CÔNG NGHỆ.....	3
2.1. Hướng giải quyết.....	3
2.2. Ngôn ngữ lập trình Java.....	3
2.3. Spring Framework.....	3
2.4. Hệ quản trị CSDL phi quan hệ MongoDB và dịch vụ MongoDB Atlas	4
2.5. Công cụ kiểm thử POSTMAN	5
2.6. Dịch vụ ZegoCloud API.....	6
Chương 3. PHÁT TRIỂN RESTful API.	7
3.1. Khái niệm cơ bản về RESTful API.....	7
3.2. Tạo dự án mới với IntelliJ và Spring Initializr	7
3.3. Cấu trúc dự án Spring Boot.....	9
3.3.1. Sơ lược về mô hình ba lớp.....	9
3.3.2. Sơ lược về mô hình MVC	10
3.3.1. Kết hợp mô hình ba lớp và mô hình MVC vào trong ứng dụng Spring Boot..	10
3.3.2. Lợi ích khi kết hợp ba mô hình	11
3.4. Sơ đồ luồng đi trong Spring Boot	12
3.5. Config MongoDB Atlas vào dự án.....	12
3.5. Thiết kế cơ sở dữ liệu.....	13

3.5.1. Collection Conversation	14
3.5.2. Collection Friendship	14
3.5.3. Collection Message.....	14
3.5.4. Collection User	15
3.6. Xây dựng các thực thể	15
3.7. Xây dựng Repository	16
3.8. Xây dựng Service	17
3.9. Xây dựng RestController	18
3.10. Kiểm thử với POSTMAN.....	19
Chương 4. CÀI ĐẶT SPRING SECURITY	21
4.1. Giới thiệu chung về Spring Security	21
4.2. Các tính năng bảo mật chính được triển khai.....	21
4.2.1. Xác thực người dùng	21
4.2.2. Mã hóa mật khẩu	21
4.2.3. Xác thực JWT	23
4.2.4. Xác thực và ủy quyền	24
4.2.5. Xử lý lỗi.....	26
4.2.6. Cơ chế bảo mật CORS:	28
4.3. Xử lý request đăng nhập và đăng ký.....	29
4.4. Kiểm thử đăng nhập và đăng ký	31
Chương 5. CÀI ĐẶT WEBSOCKET	33
5.1. Khái niệm cơ bản về WebSocket	33
5.2. Cài đặt WebSocket vào dự án Spring Boot	34
5.3. Config Spring Security cho WebSocket.	37
Chương 6. KẾT NỐI RESTful API VỚI GIAO DIỆN WEB BẰNG JAVASCRIPT	38
6.1. Vẽ giao diện.....	38

6.2. Đăng nhập.....	38
6.3. Truy vấn đến API.	40
6.4. Xử lý nhắn tin trực tiếp với WebSocket.....	41
Chương 7. TÍCH HỢP TÍNH NĂNG VIDEOCALL VÀO DỰ ÁN	43
7.1. Giới thiệu	43
7.2. Thiết kế và Cài đặt.....	43
7.2.1. Kiến trúc tổng quan	43
7.2.2. Chi tiết kỹ thuật	43
7.2.3. Quản lý trạng thái.....	45
7.3. Tính năng và Chức năng	45
7.3.1. Giao diện người dùng	45
7.3.2. Bảo mật và Quyền riêng tư	45
Chương 8. KẾT QUẢ	46
8.1. Trang chủ và đăng nhập.....	46
8.2. Tính nhắn nhắn tin với nhiều định dạng	47

DANH MỤC HÌNH ẢNH

Hình 1. Tạo dự án Spring Boot với Spring Initializr	7
Hình 2. Tạo dự án Spring Boot với IntelliJ	8
Hình 3. Thêm các Dependency vào dự án.....	9
Hình 4. Mô hình ba lớp.....	10
Hình 5. Sơ đồ luồng đi trong Spring Boot	12
Hình 6. Config thuộc tính Database vào file application.properties	12
Hình 7. Sửa đổi thuộc tính danh sách cấp quyền IP trên MongoDB Atlas	13
Hình 8. Kiểm thử bằng POSTMAN với JSON Body, phương thức POST	20
Hình 9. Kiểm thử bằng POSTMAN với Params, phương thức GET	20
Hình 10. Kiểm thử API register với POSTMAN	31
Hình 11. Kiểm thử API authenticate với POSTMAN	32
Hình 12. Cơ chế của Polling và Long Polling	33
Hình 13. Cơ chế của WebSocket.....	34
Hình 14. Giao diện trang chủ.....	46
Hình 15. Form đăng ký.....	46
Hình 16. Form đăng nhập	47
Hình 17. Tính năng nhắn tin.....	47
Hình 18. Tính năng gửi file hình ảnh, video, tệp tin,... ..	48
Hình 19. Giao diện trang cá nhân người dùng.....	48
Hình 20. Tính năng Video Call	49
Hình 21. Tính năng nhắn tin giữa cuộc gọi	49

DANH MỤC TỪ KHÓA – VIẾT TẮT

RESTful API	(Representational State Transfer API) là một kiểu kiến trúc thiết kế API trong phát triển phần mềm dựa trên các nguyên tắc của REST
Framework	Tập hợp các quy tắc và hướng dẫn được tổ chức hệ thống
Annotation	Cách để cung cấp metadata (dữ liệu mô tả) cho các thành phần của mã nguồn
Dependency	Một thành phần hoặc tài nguyên mà một phần của mã nguồn phụ thuộc vào để hoạt động đúng cách
Module	Một phần của mã nguồn được tổ chức thành một đơn vị độc lập
Starter	Một dự án, một mẫu, hoặc một tập hợp các cài đặt ban đầu có thể được sử dụng để bắt đầu một ứng dụng mới một cách nhanh chóng
CSRF	(Cross-Site Request Forgery) một kiểu tấn công mạng thông qua việc gửi các yêu cầu HTTP không mong muốn từ một trang web khác
JSON	(JavaScript Object Notation) một định dạng dữ liệu dựa trên văn bản phổ biến dựa trên cú pháp dễ đọc và dễ hiểu
Atomicity	Nguyên tắc tóm gọn
Consistency	Tính nhất quán
Isolation	Tính cô lập
Durability	Tính bền vững
XML	(eXtensible Markup Language) một ngôn ngữ đánh dấu dựa trên văn bản được sử dụng để lưu trữ và truyền tải dữ liệu có cấu trúc
WebRTC	(Web Real-Time Communication) một công nghệ tiêu chuẩn cho việc thiết lập trực tuyến truyền thông và truyền dữ liệu trong các ứng dụng web mà không cần cài đặt phần mềm bổ sung hoặc plugins
JWT	(JSON Web Token) một tiêu chuẩn mở để định dạng và truyền tải thông tin xác thực giữa các bên trong một ứng dụng web
HTTP	(Hypertext Transfer Protocol) một giao thức truyền tải dữ liệu giữa máy khách và máy chủ trên Internet
Service	Đánh dấu một lớp là tầng Service, phục vụ các logic nghiệp vụ.
Repository	Đánh dấu một lớp Là tầng Repository, phục vụ truy xuất dữ liệu.
application.properties	Nơi chứa cấu hình mặc định của ứng dụng Spring
React	Một thư viện JavaScript front-end mã nguồn mở và miễn phí để xây dựng giao diện người dùng dựa trên các thành phần UI riêng lẻ. Nó được phát triển và duy trì bởi Meta và cộng đồng các nhà phát triển và công ty cá nhân.
Angular	Angular là một JavaScript framework dùng để viết giao diện web (Front-end), được phát triển bởi Google.
Uniform Interface	Một trong những ràng buộc (constraints) cơ bản của kiến trúc REST (Representational State Transfer). Được định nghĩa bởi Roy Fielding trong luận văn tiến sĩ của mình, Uniform Interface yêu cầu rằng các thành phần của hệ thống REST phải sử dụng một giao diện thống nhất để truy cập và quản lý các tài nguyên (resources).
Uniform Resource Identifier	Một chuỗi ký tự để xác định một tài nguyên trên Internet.

LỜI MỞ ĐẦU

Kỷ nguyên số hóa hiện nay đang chứng kiến sự bùng nổ của công nghệ, tạo ra những thay đổi mạnh mẽ trong mọi lĩnh vực của xã hội. Đặc biệt, nhu cầu giao tiếp và kết nối giữa con người ngày càng cao, thúc đẩy sự phát triển không ngừng của các phương thức liên lạc mới. Những phương thức này đã giúp con người dễ dàng tiếp cận và kết nối với nhau, vượt qua mọi rào cản về không gian và thời gian.

Đại dịch Covid-19 đã làm tăng tầm quan trọng của việc giao tiếp trực tuyến. Khi việc gặp gỡ trực tiếp trở nên hạn chế, nhắn tin và tương tác qua mạng đã trở thành phương tiện chính cho các cuộc họp, lớp học trực tuyến, thăm khám y tế và cả các hoạt động xã hội. Những ứng dụng nhắn tin, gọi điện video đã trở thành công cụ không thể thiếu để duy trì kết nối giữa con người trong thời kỳ giãn cách xã hội.

Nhận thức được tầm quan trọng của lĩnh vực này, nhóm chúng em đã quyết định chọn đề tài “Nghiên cứu và phát triển ứng dụng nhắn tin và gọi điện trực tuyến”. Mục tiêu của chúng em là tìm hiểu và phát triển ứng dụng nhắn tin, nhằm góp phần nâng cao hiệu quả giao tiếp trong thời đại số hóa.

Chương 1. TỔNG QUAN VỀ ĐỀ TÀI

1.1. Giới thiệu về ứng dụng nhắn tin và gọi điện trực tuyến

Chúng ta đang sống trong thời đại kỹ thuật số, nơi mà việc giao tiếp nhanh chóng và hiệu quả trở nên cực kỳ quan trọng. Các ứng dụng nhắn tin theo thời gian như như Telegram, What's App hay Zalo cũng ra đời từ nhu cầu này

Tính năng nhắn tin thời gian thực cho phép người dùng gửi và nhận tin nhắn tức thì, tạo ra một trải nghiệm giao tiếp mượt mà và liền mạch. Tính năng gọi điện trực tuyến cho phép người tham gia cuộc gọi có thể nhìn thấy mặt của nhau và hơn thế nữa. Tuy nhiên, việc xây dựng các tính năng này đòi hỏi đội ngũ lập trình phải biết các giao thức mạng như WebSocket, WebRTC, ... cách bảo mật ứng dụng sao cho tránh lộ lọt thông tin giảm thiểu đến mức độ thấp nhất.

Trong đề tài này, chúng ta sẽ tìm hiểu cách xây dựng tính năng nhắn tin thời gian thực cho ứng dụng chat, từ việc thiết lập cơ sở hạ tầng cho đến việc triển khai và kiểm thử. Chúng ta cũng sẽ tìm hiểu cách giải quyết các thách thức và vấn đề phát sinh trong quá trình xây dựng, để tạo ra một ứng dụng chat thời gian thực hiệu quả và mạnh mẽ.

1.2. Yêu cầu thực hiện

Nhóm sẽ xây dựng ứng dụng với các tính năng cơ bản dưới đây.

Tìm kiếm bạn bè, kết bạn:

- + Tìm kiếm theo tên: Người dùng có thể nhập tên, biệt danh hoặc email của người ta muốn tìm kiếm.
- + Yêu cầu kết bạn: Người dùng có thể gửi yêu cầu kết bạn đến người khác. Khi yêu cầu được chấp nhận, hai người sẽ trở thành bạn bè trên ứng dụng.

Trò chuyện theo thời gian thực:

- + Nhắn tin cá nhân: Người dùng có thể nhắn tin riêng tư với nhau.
- + Chia sẻ hình ảnh, video và tệp tin: Người dùng có thể chia sẻ hình ảnh, video và tệp tin với bạn bè trong tin nhắn cá nhân và nhóm trò chuyện.

Gọi điện trực tuyến:

- + Gọi thoại: Người dùng có thể gọi điện thoại trực tuyến cho bạn bè trên ứng dụng.
- + Gọi video: Người dùng có thể gọi video trực tuyến cho bạn bè trên ứng dụng.
- + Chất lượng cuộc gọi: Ứng dụng sẽ đảm bảo chất lượng cuộc gọi tốt nhất cho người dùng.

Các tính năng đi kèm:

- + Chỉnh sửa thông tin cá nhân: Người dùng có thể chỉnh sửa thông tin cá nhân như tên, ảnh đại diện, ảnh bìa, v.v.
- + Chỉnh sửa biệt danh: Người dùng có thể chọn biệt danh để sử dụng trên ứng dụng.
- + Cài đặt bảo mật: Người dùng có thể cài đặt các tùy chọn bảo mật để kiểm soát quyền riêng tư của họ trên ứng dụng.
- + Lịch sử trò chuyện: Người dùng có thể xem lại lịch sử trò chuyện của họ với bạn bè.
- + Thông báo: Người dùng sẽ nhận được thông báo về các hoạt động mới trên ứng dụng, chẳng hạn như tin nhắn mới, yêu cầu kết bạn, v.v.

Trang đăng nhập:

- + Đăng ký: Người dùng mới có thể tạo tài khoản bằng cách nhập thông tin cá nhân như tên, email, mật khẩu, v.v.
- + Đăng nhập: Người dùng đã có tài khoản có thể đăng nhập bằng email và mật khẩu của họ.
- + Quên mật khẩu: Người dùng có thể đặt lại mật khẩu nếu họ quên.

Chương 2: THIẾT KẾ VÀ LỰA CHỌN CÔNG NGHỆ

2.1. Hướng giải quyết

Để xây dựng một ứng dụng nhắn tin với các yêu cầu trong phần 1.2, ta cần xây dựng một RESTful API để xử lý các tác vụ đối với người dùng và cơ sở dữ liệu, và xây dựng giao diện Web với các ngôn ngữ HTML, CSS và JavaScript.

Đồng thời, ta sử dụng công nghệ WebSocket để xây dựng ứng dụng theo thời gian thực và Spring Security để bảo mật trang Web để tránh rò rỉ dữ liệu người dùng

Các phần tiếp theo đây sẽ nói chi tiết hơn về các công nghệ mà nhóm lựa chọn và sử dụng.

2.2. Ngôn ngữ lập trình Java

Trong dự án này, nhóm lựa chọn ngôn ngữ Java vì đây là một ngôn ngữ có cú pháp rất chặt chẽ, hiệu năng tốt và linh hoạt, mang lại nhiều ưu điểm khi xây dựng ứng dụng Web. Dưới đây là các ưu điểm của Java:

- + **Đa nền tảng:** Java chạy được trên nhiều hệ điều hành, giúp ứng dụng hoạt động đa dạng các nền tảng khác nhau.

- + **Bảo mật:** Java có độ bảo mật cao, giảm thiểu rủi ro tấn công.

- + **Hỗ trợ thư viện phong phú:** Phát triển nhanh chóng, hiệu quả với nhiều thư viện và framework sẵn có như Spring hay Hibernate ...

- + **Hiệu suất cao:** Xử lý tốt lượng truy cập lớn, phù hợp cho ứng dụng nhắn tin thời gian thực.

- + **Quản lý bộ nhớ tự động:** Giảm lỗi bộ nhớ, tăng hiệu suất ứng dụng.

2.3. Spring Framework

Để xây dựng RESTful API, nhóm sử dụng Spring Framework. Đây là một Framework mạnh mẽ, hỗ trợ tốt trong việc xây dựng RESTful API với việc cung cấp các Annotation và cấu hình đơn giản để xây dựng RESTful API, giúp tiết kiệm thời gian và công sức cho việc phát triển. Ngoài ra, Spring Framework có những tính năng mạnh mẽ khác như

- + **Quản lý Dependency:** Spring Boot giúp quản lý Dependency hiệu quả, tự động tải về và cấu hình các thư viện cần thiết cho ứng dụng.

+ Tích hợp nhiều tính năng: Spring tích hợp nhiều tính năng hữu ích như bảo mật, truy cập dữ liệu, giao dịch, xử lý lỗi, ... giúp đơn giản hóa việc phát triển ứng dụng.

Các module được sử dụng trong dự án này ta có thể kể đến:

+ **Spring Boot:** Spring Boot là một dự án của Spring giúp đơn giản hóa việc cấu hình và triển khai ứng dụng Spring. Nó cung cấp các “starter” tự động cấu hình các thư viện phụ thuộc, giúp ta tập trung vào việc xây dựng ứng dụng mà không phải lo lắng về việc cấu hình.

+ **Spring Security:** Spring Security là một framework mạnh mẽ và linh hoạt giúp quản lý xác thực và ủy quyền cho ứng dụng Java. Nó cung cấp các giải pháp bảo mật toàn diện cho ứng dụng web, từ xác thực, ủy quyền, chống tấn công CSRF, đến bảo mật REST API.

+ **Spring WebSocket:** Spring WebSocket là một mô-đun của Spring cung cấp hỗ trợ cho giao tiếp WebSocket trong ứng dụng web. WebSocket là một giao thức cung cấp kết nối hai chiều giữa máy khách và máy chủ, cho phép truyền dữ liệu thời gian thực. Điều này rất hữu ích cho các ứng dụng chat, khi ta muốn cập nhật thông tin ngay lập tức mà không cần tải lại trang. Spring WebSocket giúp đơn giản hóa việc xây dựng những ứng dụng như vậy.

2.4. Hệ quản trị CSDL phi quan hệ MongoDB và dịch vụ MongoDB Atlas

Là một ứng dụng nhắn tin, việc sử dụng cơ sở dữ liệu quan hệ để lưu các cuộc trò chuyện là không hợp lý, vì hằng ngày tin nhắn được gửi đi và lưu vào cơ sở dữ liệu rất nhiều, việc truy vấn với cơ sở dữ liệu quan hệ sẽ khá chậm và dễ dàng rò rỉ thông tin hơn, không phù hợp với dự án. Do vậy, nhóm quyết định sử dụng cơ sở dữ liệu phi quan hệ. MongoDB rất phù hợp vì RESTful API sau khi xây dựng sẽ trả về JSON, tạo nên sự nhất quán thì request và response dữ liệu.

MongoDB là một hệ quản trị cơ sở dữ liệu (CSDL) phi quan hệ, dựa trên mô hình tài liệu. Điều này có nghĩa là dữ liệu được lưu trữ dưới dạng các tài liệu BSON có thể linh hoạt, không cần tuân thủ một cấu trúc cố định như trong các CSDL quan hệ truyền thống.

Nói sơ lược về BSON (Binary JSON) là một định dạng dữ liệu được sử dụng để lưu trữ và trao đổi dữ liệu giữa các ứng dụng. Nó dựa trên định dạng JSON phổ biến, nhưng được tối ưu hóa để lưu trữ dữ liệu nhị phân hiệu quả hơn.

Dưới đây là một số đặc điểm chính của MongoDB:

- + Mô hình dữ liệu linh hoạt: MongoDB cho phép lưu trữ dữ liệu dưới dạng các tài liệu JSON, không yêu cầu cấu trúc cố định như trong CSDL quan hệ. Điều này giúp phát triển ứng dụng một cách linh hoạt và thích ứng dễ dàng với các thay đổi cấu trúc dữ liệu.

- + Tính mở rộng cao: MongoDB hỗ trợ tính mở rộng ngang, cho phép mở rộng hệ thống bằng cách thêm các node vào cụm MongoDB một cách dễ dàng, giúp tăng khả năng chịu tải của ứng dụng.

- + Tính nhất quán: MongoDB hỗ trợ các giao thức nhất quán như ACID (Atomicity, Consistency, Isolation, Durability) cho các giao dịch, cung cấp sự tin cậy và độ tin cậy cao đối với dữ liệu.

- + Tích hợp tốt với ứng dụng realtime: Do tính linh hoạt và khả năng mở rộng của MongoDB, nó rất phù hợp cho việc xây dựng ứng dụng nhắn tin realtime. Các tính năng như tài liệu JSON linh hoạt và ghi chép (replication) và phân tán (sharding) giúp cho việc xử lý dữ liệu realtime một cách hiệu quả.

- + Phát triển nhanh chóng: MongoDB có thể được triển khai nhanh chóng và dễ dàng cấu hình, giúp tăng tốc độ phát triển ứng dụng.

Vì lý do này, MongoDB là một lựa chọn để nhóm xây dựng các ứng dụng nhắn tin realtime, với khả năng mở rộng, tính nhất quán và tích hợp tốt với các công nghệ realtime khác như WebSocket hoặc các dịch vụ nhắn tin realtime khác.

2.5. Công cụ kiểm thử POSTMAN

Để kiểm thử RESTful API, nhóm sử dụng công cụ POSTMAN, lý do như sau

Lý do chọn Postman:

- + Dễ sử dụng: Postman có giao diện trực quan, dễ dàng sử dụng cho cả người mới bắt đầu.

- + Hỗ trợ đa dạng: Postman hỗ trợ nhiều phương thức HTTP như GET, POST, PUT, DELETE, ... và nhiều định dạng dữ liệu như JSON, XML, ...

- + Tính năng mạnh mẽ: Postman cung cấp nhiều tính năng hữu ích cho việc kiểm thử API như gửi yêu cầu, nhận phản hồi, kiểm tra dữ liệu, lưu trữ collections, ...

+ Hỗ trợ cộng đồng: Postman có cộng đồng người dùng lớn và nhiều tài liệu hướng dẫn, giúp dễ dàng tìm kiếm hỗ trợ khi gặp vấn đề.

2.6. Dịch vụ ZegoCloud API

Vì nguồn lực có hạn, nhóm không thể xây dựng phương thức WebRTC để gọi điện trực tuyến, do vậy, nhóm sẽ sử dụng bên thứ ba, cụ thể là ZegoCloud.

ZegoCloud API là một tập hợp các API cung cấp các tính năng giao tiếp thời gian thực như trò chuyện thoại, trò chuyện video, chia sẻ màn hình, ... cho các ứng dụng di động và web. ZegoCloud API được thiết kế để giúp các nhà phát triển dễ dàng tích hợp các tính năng giao tiếp thời gian thực vào ứng dụng của họ một cách nhanh chóng và hiệu quả.

Các tính năng chính của ZegoCloud API:

+ Trò chuyện thoại: Hỗ trợ trò chuyện thoại một đối một và nhóm với chất lượng âm thanh cao.

+ Trò chuyện video: Hỗ trợ trò chuyện video một đối một và nhóm với chất lượng video cao, bao gồm HD và 4K.

+ Chia sẻ màn hình: Hỗ trợ chia sẻ màn hình của người dùng với các thành viên khác trong cuộc trò chuyện.

+ Phát trực tiếp: Hỗ trợ phát trực tiếp video và âm thanh đến nhiều người xem.

Chương 3. PHÁT TRIỂN RESTful API.

3.1. Khái niệm cơ bản về RESTful API

RESTful API (Representational State Transfer) là một kiểu kiến trúc cho việc xây dựng các dịch vụ web. Dưới đây là một số khái niệm cơ bản về RESTful API:

- + Resource-Based: Trong REST, dữ liệu và chức năng của hệ thống được xem như là các "resource" và được truy cập thông qua URI (Uniform Resource Identifier).

- + Client-Server: Client (front-end) và Server (back-end) hoạt động độc lập và có thể phát triển một cách riêng biệt.

- + Uniform Interface: REST sử dụng một giao diện thống nhất, bao gồm các phương thức HTTP như GET, POST, PUT, DELETE.

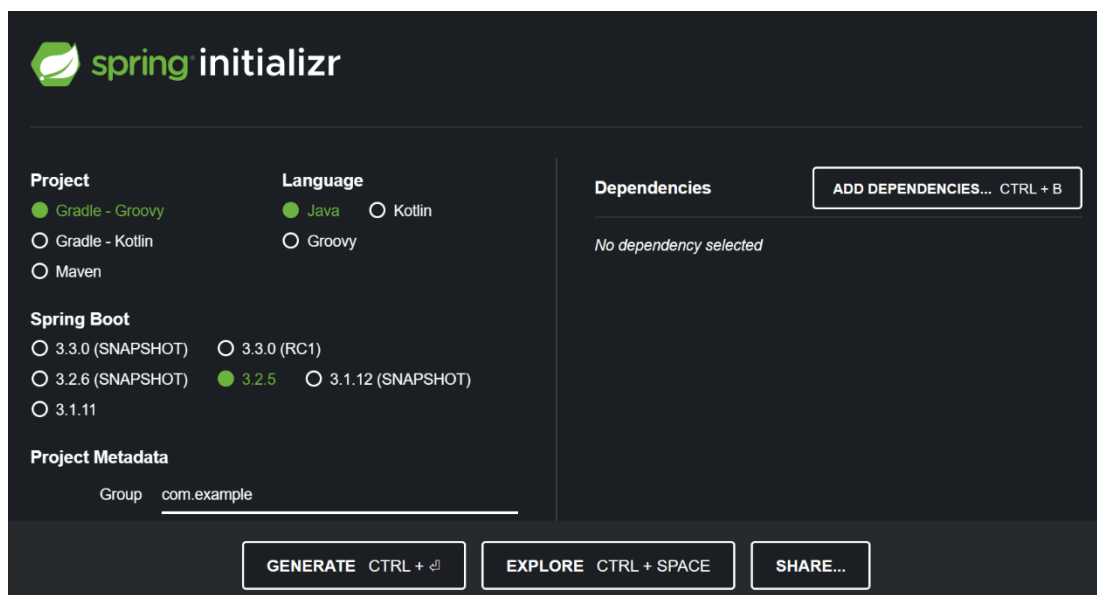
- + Layered System: Hỗ trợ sự tách biệt giữa các thành phần trong hệ thống, cho phép các thành phần hoạt động độc lập.

RESTful API thường trả về dữ liệu ở định dạng JSON hoặc XML, giúp cho việc trao đổi dữ liệu giữa client và server trở nên dễ dàng và linh hoạt.

3.2. Tạo dự án mới với IntelliJ và Spring Initializr

Có hai cách để tạo dự án Spring Boot mới:

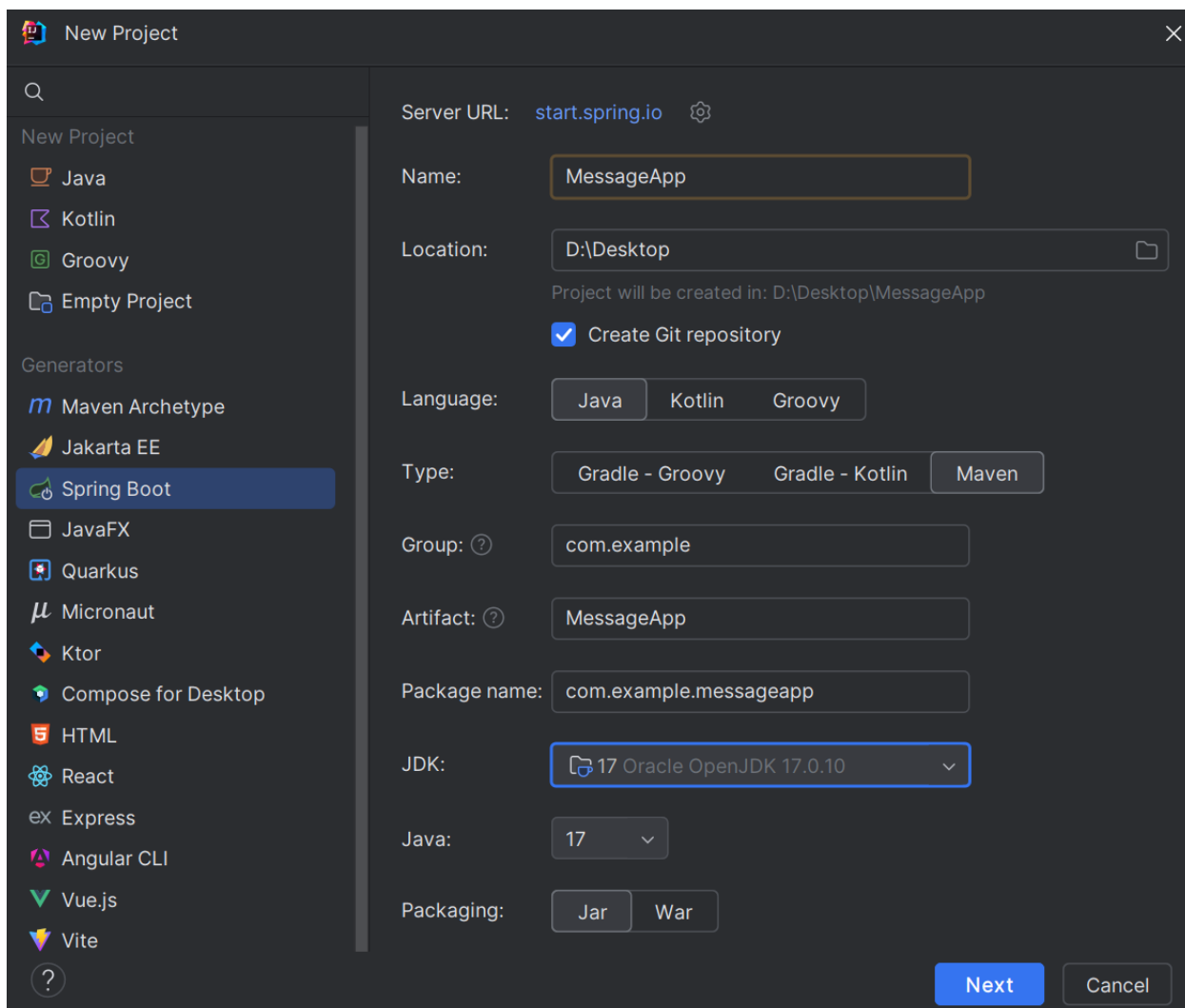
Cách 1: *Truy cập vào trang start.spring.io và tùy chỉnh các thuộc tính, cài đặt các dependency phù hợp với dự án. Sau đó, ấn Generate để tải xuống dự án*

The image shows the Spring Initializr web interface. It has a dark theme. At the top left is the Spring logo and the text 'spring initializr'. Below this, there are three main sections: 'Project', 'Language', and 'Spring Boot'. The 'Project' section has three radio buttons: 'Gradle - Groovy' (selected), 'Gradle - Kotlin', and 'Maven'. The 'Language' section has three radio buttons: 'Java' (selected), 'Kotlin', and 'Groovy'. The 'Spring Boot' section has five radio buttons: '3.3.0 (SNAPSHOT)', '3.3.0 (RC1)', '3.2.6 (SNAPSHOT)', '3.2.5' (selected), and '3.1.12 (SNAPSHOT)'. Below these is the 'Project Metadata' section with a 'Group' field containing 'com.example'. To the right of these sections is a 'Dependencies' section with a button 'ADD DEPENDENCIES... CTRL + B' and the text 'No dependency selected'. At the bottom, there are three buttons: 'GENERATE CTRL + G', 'EXPLORE CTRL + SPACE', and 'SHARE...'.

Project	Language	Spring Boot	Project Metadata	Dependencies
<input checked="" type="radio"/> Gradle - Groovy	<input checked="" type="radio"/> Java	<input checked="" type="radio"/> 3.2.5	Group: com.example	No dependency selected
<input type="radio"/> Gradle - Kotlin	<input type="radio"/> Kotlin	<input type="radio"/> 3.3.0 (SNAPSHOT)		
<input type="radio"/> Maven	<input type="radio"/> Groovy	<input type="radio"/> 3.3.0 (RC1)		
		<input type="radio"/> 3.2.6 (SNAPSHOT)		
		<input type="radio"/> 3.1.12 (SNAPSHOT)		

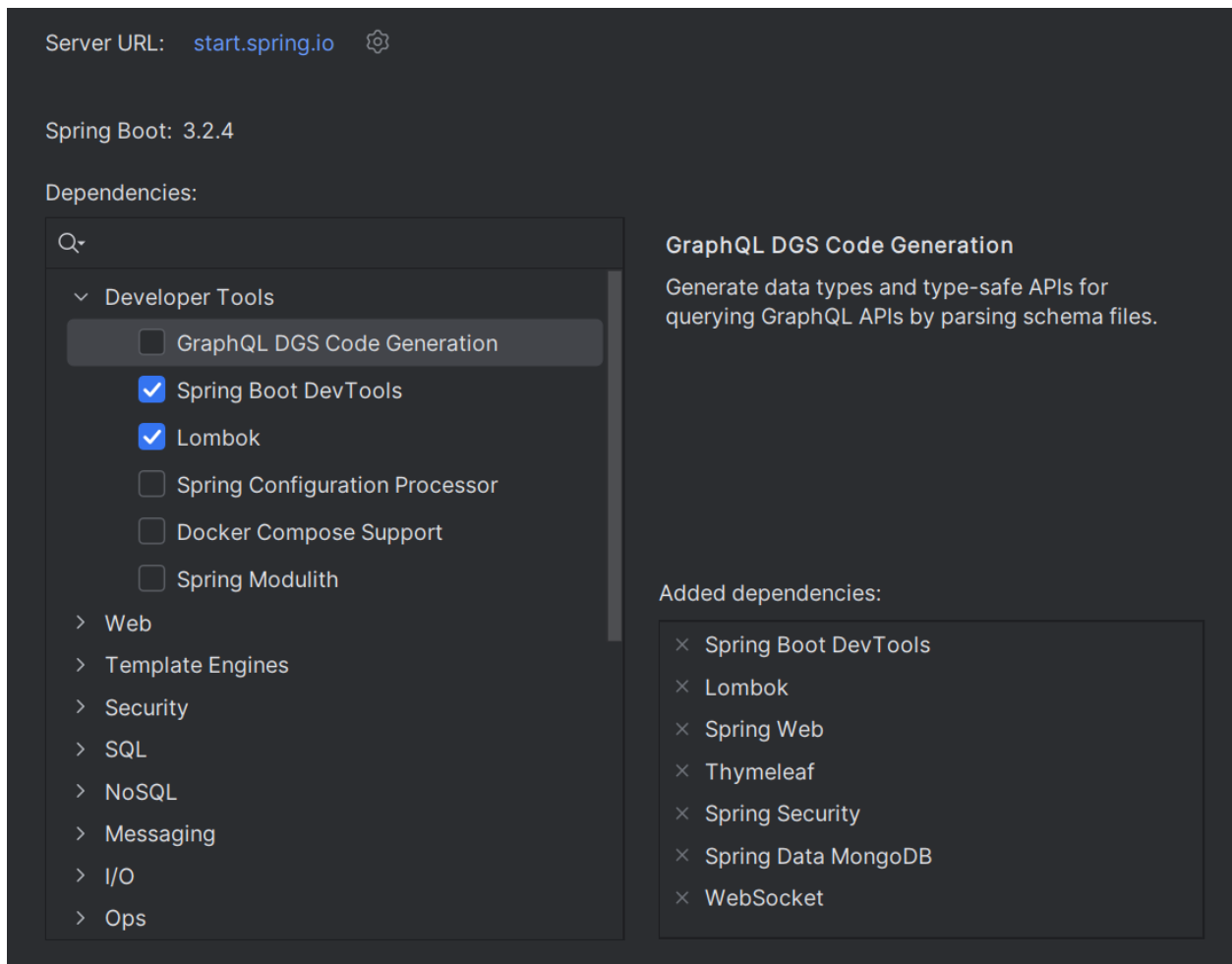
Hình 1. Tạo dự án Spring Boot với Spring Initializr

Cách 2: Tạo dự án mới trên IntelliJ bằng cách chọn New Project → Spring Boot. Vì ở đây, nhóm sử dụng IntelliJ để tạo dự án mới nên nhóm sẽ tùy chỉnh trong IntelliJ như Hình 2.



Hình 2. Tạo dự án Spring Boot với IntelliJ

Sau khi tùy chỉnh các thuộc tính, chúng ta sẽ cài đặt các dependency cần thiết (Chúng ta có thể thay đổi các dependency này trong dự án) như Hình 3.



Hình 3. Thêm các Dependency vào dự án

Sau khi tạo xong, IntelliJ sẽ tự động tải xuống các Dependency, khi này, ta có thể bắt đầu chạy và lập trình dự án.

3.3. Cấu trúc dự án Spring Boot

Về phần này, Spring khá linh hoạt khi chúng ta có thể tổ chức dự án theo nhiều cách khác nhau như là tổ chức theo tính năng, tổ chức theo các mô hình phổ biến như mô hình ba lớp, mô hình MVC ...

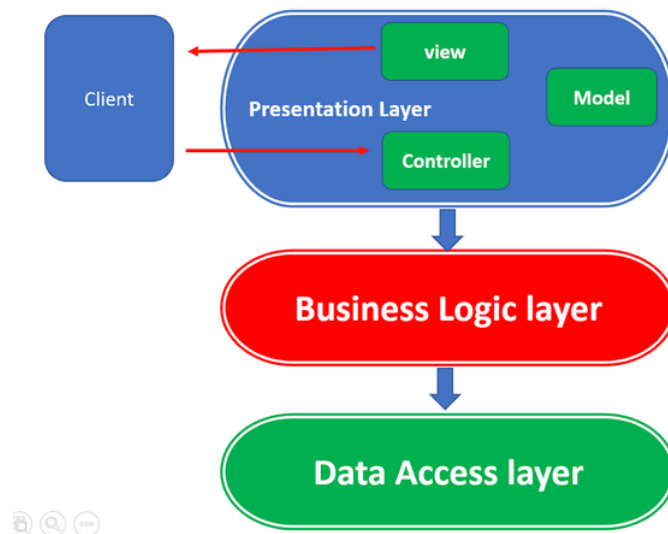
Trong dự án này, nhóm sử dụng kết hợp giữa mô hình MVC và mô hình ba lớp, đây là mô hình phổ biến mà hầu như các dự án Java Spring từ lớn đến nhỏ đều sử dụng

3.3.1. Sơ lược về mô hình ba lớp

Đây là mô hình tổ chức source code rất phổ biến trong Spring Boot. Cụ thể, ứng dụng được chia làm 3 tầng (tier hoặc layer) như sau:

+ **Presentation layer:** Tầng này tương tác với người dùng, bằng View, Controller (trong MVC) hoặc API (nếu có).

- + **Business logic layer:** Chứa toàn bộ logic của chương trình, các đa số code nằm ở đây
- + **Data access layer:** Tương tác với database, trả về kết quả cho tầng business logic



Hình 4. Mô hình ba lớp

Trong Spring Boot, thì có một số thành phần đại diện cho từng lớp như thành phần Service chứa các business logic code hay Repository đại diện cho tầng data access

3.3.2. Sơ lược về mô hình MVC

Mô hình này đại diện cho phần Presentation của mô hình ba lớp ở phần 3.3.1, gồm ba thành phần:

- + **Controller:** Trả về View (có chứa data sẵn, dạng trang HTML), hoặc Model thể hiện dưới dạng API cho View (View viết riêng bằng React, Vue, hoặc Angular).

- + **Service:** Chứa các code tính toán, xử lý. Khi Controller yêu cầu, thì Service tương ứng sẽ tiếp nhận và cho ra dữ liệu trả cho Controller (trả về Model). Controller sẽ gửi về View như trên.

- + **Repository:** Service còn có thể tương tác với service khác, hoặc dùng Repository để gọi DB. Repository là thẳng trực tiếp tương tác, đọc ghi dữ liệu trong DB và trả cho service.

3.3.1. Kết hợp mô hình ba lớp và mô hình MVC vào trong ứng dụng Spring Boot.

Kết hợp hai mô hình lại, chúng ta có được ứng dụng Spring Boot được cấu trúc bởi các thành phần chính sau:

Controller:

+ Chịu trách nhiệm xử lý các yêu cầu từ người dùng và trả về dữ liệu dưới dạng View (HTML) hoặc Model (dạng API).

Service:

- + Chứa các logic xử lý nghiệp vụ, thực hiện các tính toán và thao tác dữ liệu.
- + Tương tác với Repository để truy cập và lưu trữ dữ liệu trong cơ sở dữ liệu.
- + Cung cấp dữ liệu cho Controller theo yêu cầu.

Repository:

- + Trực tiếp tương tác với cơ sở dữ liệu, thực hiện các truy vấn INSERT, UPDATE, DELETE, SELECT.
- + Cung cấp dữ liệu cho Service khi cần thiết.

Trong ứng dụng Spring, khi cấu trúc Database, ta hay đặt tên thư mục là model, nhưng trên thực chất, chúng đóng vai trò là các Entity (Thực thể). Thực chất, model trong Spring là các đối tượng được Service tính toán xong trả về cho Controller.

View: Trình bày dữ liệu cho người dùng dưới dạng giao diện. Có hai loại chính:

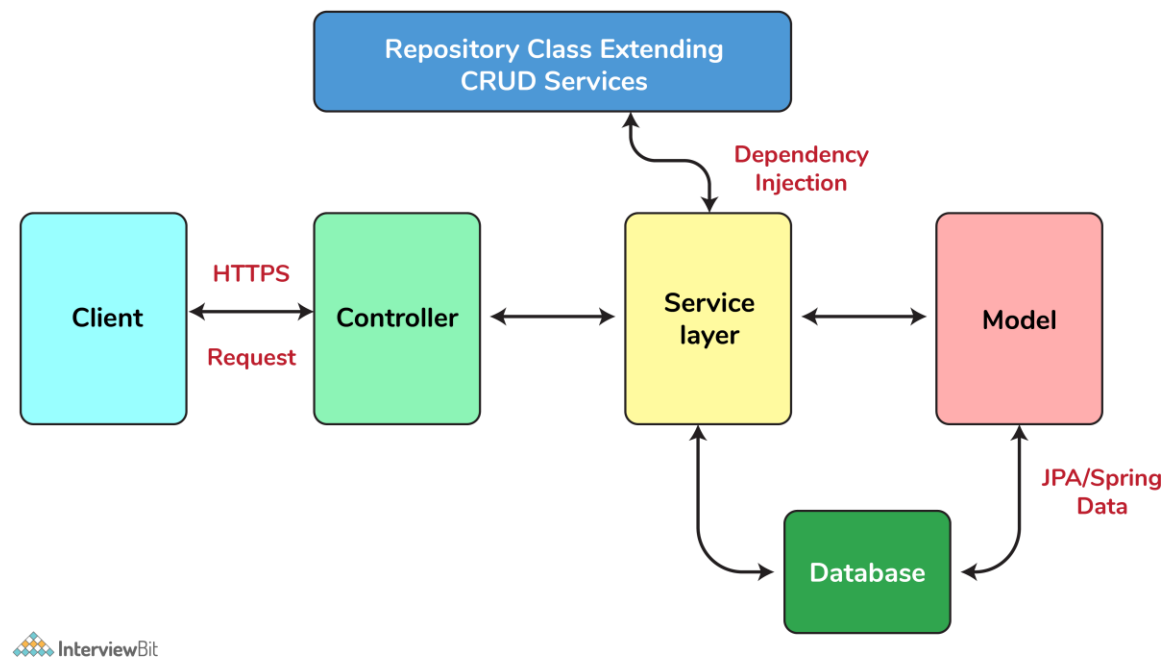
- + **View truyền thống:** HTML được nhúng trực tiếp trong ứng dụng Spring Boot, sử dụng JSP hoặc Thymeleaf để hiển thị dữ liệu.
- + **View tách rời:** Viết riêng bằng React, Angular, v.v., giao tiếp với Controller thông qua API.

3.3.2. Lợi ích khi kết hợp ba mô hình

- + Kiến trúc rõ ràng, dễ hiểu, dễ bảo trì.
- + Phân chia trách nhiệm hợp lý, giúp code dễ quản lý và tái sử dụng.
- + Tăng khả năng mở rộng, thích ứng với các thay đổi về yêu cầu nghiệp vụ.
- + Hỗ trợ phát triển các ứng dụng web hiện đại, sử dụng các framework JavaScript phổ biến.

3.4. Sơ đồ luồng đi trong Spring Boot

Spring Boot Flow Architecture



Hình 5. Sơ đồ luồng đi trong Spring Boot

3.5. Config MongoDB Atlas vào dự án

Để thuận tiện hơn trong quá trình làm việc nhóm, thì việc sử dụng database có sẵn trong máy là khá bất tiện, do vậy, nhóm đã đăng ký dịch vụ MongoDB Atlas và cài đặt vào dự án vào file application.properties, với các thuộc tính như sau.

```
application.properties × Message.java MessageRepository.java User.java
1 spring.data.mongodb.uri=mongodb+srv://vinhnqu64cntt:12345@cluster0.nra8a56.moi
2 spring.data.mongodb.database=message_app
```

Hình 6. Config thuộc tính Database vào file application.properties

Trong đó

spring.data.mongodb.uri: Đường liên kết truy cập vào database, với các thông tin như mật khẩu, tên database ... như dưới đây

mongodb+srv://vinhnqu64cntt:<Mật_khẩu>@cluster0.nra8a56.mongodb.net/<tên database>?retryWrites=true&w=majority&appName=Cluster0

spring.data.mongodb.database: Tên database ta muốn truy cập.

Các thuộc tính trên nằm trong gói Spring Data MongoDB, hãy đảm bảo gói này được cài đặt trước khi kết nối database

MongoDB Atlas có một tính năng khá bảo mật là chặn các IP truy cập trái phép (Là những IP người dùng chưa cấp quyền), vì vậy để sử dụng được, ta cần phải thêm IP server vào.

Để thuận tiện cho việc testing và deploy ứng dụng, nhóm đã sửa đổi thuộc tính IP Access List Entry là 0.0.0.0/0. Điều này đồng nghĩa với việc tất cả các máy tính miễn có Internet đều có thể truy cập.

Edit IP Access List Entry ✕

Atlas only allows client connections to a cluster from entries in the project's IP Access List. Each entry should either be a single IP address or a CIDR-notated range of addresses. [Learn more.](#)

ADD CURRENT IP ADDRESS

Access List Entry:

Comment:

Cancel

Confirm

Hình 7. Sửa đổi thuộc tính danh sách cấp quyền IP trên MongoDB Atlas

3.5. Thiết kế cơ sở dữ liệu

Vì là một cơ sở dữ liệu phi quan hệ, nên các bảng hoàn toàn tách biệt với nhau. Database dưới đây được chia làm 6 collection, bao gồm

- + Conservation: Cuộc trò chuyện
- + Friendship: Quan hệ bạn bè
- + Message: Tin nhắn
- + User: Thông tin người dùng, bao gồm mật khẩu, ...

Căn cứ vào tình hình thực tế của dự án, và nhiều vấn đề khác, nhóm đã chọn cách thiết kế như trên, nhằm mục đích dễ dàng truy vấn và mở rộng thêm các tính năng mà

không cần phải thay đổi database, đảm bảo tính toàn vẹn dữ liệu và hạn chế việc trùng lặp thông tin.

Các bảng đều có trường id và để kết nối các bảng lại với nhau, ta sẽ truy vấn thông qua trường id giữa các bảng. Do vậy để ràng buộc dữ liệu tránh bị trùng lặp, ta phải ràng buộc dữ liệu ngay từ phần Service hoặc FrontEnd (Cảnh báo người dùng khi nhập sai thông tin).

3.5.1. Collection Conversation

Mỗi document trong collection này sẽ biểu diễn một cuộc trò chuyện. Mỗi document sẽ có các trường sau:

- + **id**: định danh duy nhất cho cuộc trò chuyện.
- + **conservationName**: tên của cuộc trò chuyện.
- + **participantId**: danh sách các ID của người tham gia cuộc trò chuyện.

3.5.2. Collection Friendship

Mỗi document trong collection này sẽ biểu diễn một mối quan hệ giữa hai người dùng. Mỗi document sẽ có các trường sau:

- + **id**: định danh duy nhất cho mối quan hệ.
- + **senderId**: ID của người gửi lời mời kết bạn.
- + **receiverId**: ID của người nhận lời mời kết bạn.
- + **status**: trạng thái của mối quan hệ. Có thể là 'PENDING' nếu lời mời kết bạn đang chờ được chấp nhận, hoặc 'ACCEPTED' nếu lời mời đã được chấp nhận.
- + **timestamp**: thời gian lời mời được gửi.

3.5.3. Collection Message

Mỗi document trong collection này sẽ biểu diễn một tin nhắn trong một cuộc trò chuyện. Mỗi document sẽ có các trường sau:

- + **id**: định danh duy nhất cho tin nhắn.
- + **conservationId**: ID của cuộc trò chuyện mà tin nhắn thuộc về.
- + **userId**: ID của người gửi tin nhắn.

- + **type**: loại tin nhắn. Có thể là 'TEXT', 'IMAGE', 'VIDEO', 'AUDIO', hoặc 'FILE'.
- + **content**: nội dung của tin nhắn.
- + **timestamp**: thời gian tin nhắn được gửi.

3.5.4. Collection User

Mỗi document trong collection này sẽ biểu diễn một người dùng. Mỗi document sẽ có các trường sau:

- + **id**: định danh duy nhất cho người dùng.
- + **fullName**: tên đầy đủ của người dùng.
- + **gender**: giới tính của người dùng.
- + **dateOfBirth**: ngày sinh của người dùng.
- + **email**: email của người dùng.
- + **avatar**: đường dẫn tới ảnh đại diện của người dùng.
- + **password**: mật khẩu của người dùng.

3.6. Xây dựng các thực thể

Sau khi đã có bảng thiết kế cơ sở dữ liệu hoàn chỉnh, nhóm sẽ cài đặt các thực thể này vào trong dự án.

Ta sẽ tạo một package mang tên là **Entity** vào địa chỉ gói **org.vinhveer.message**, với nhiệm vụ là lưu trữ các thực thể database có trong dự án.

Ví dụ với Message, ta sẽ cài đặt như sau:

```
@Document
@Data
@AllArgsConstructor
@NoArgsConstructor
public class Message {
    @Id
    private String id;
    private String conservationId;
    private String userId;
    private Type type;
    private String content;
    private long timestamp;

    public enum Type { TEXT, IMAGE, VIDEO, AUDIO, FILE }
}
```

Mã nguồn trên đang định nghĩa một thực thể (entity) trong Java và MongoDB tên là "Message". Cụ thể, thực thể này được tạo thông qua việc sử dụng các annotations của Spring và Lombok.

@Document: Annotation này từ Spring Data MongoDB, dùng để đánh dấu class này là một document trong MongoDB.

@Data: Annotation này từ thư viện Lombok, tự động tạo các getters, setters, equals, hashCode và toString cho class.

@AllArgsConstructor: Annotation này từ Lombok, tự động tạo constructor với tất cả các trường làm tham số.

@NoArgsConstructor: Annotation này từ Lombok, tự động tạo constructor không tham số.

@Id: Annotation này từ Spring, đánh dấu trường "id" là khóa chính của document trong MongoDB.

Tương tự với các thực thể khác

Sau khi cài đặt xong, chúng ta sẽ chạy dự án. Khi này, MongoDB đã tạo sẵn các Collection.

3.7. Xây dựng Repository

Sau khi xây dựng các thực thể xong, chúng ta sẽ xây dựng lớp Repository.

Ví dụ với Message, ta sẽ cài đặt như sau:

```
@Repository
public interface MessageRepository extends MongoRepository<Message, String> {
    List<Message> findByConservationId(String conservationId);
}
```

Lớp Repository được tạo ra để thực hiện các thao tác tương tác với cơ sở dữ liệu, bao gồm việc truy vấn, thêm, xóa và cập nhật dữ liệu. Trong trường hợp này, MessageRepository được tạo ra để quản lý các thao tác liên quan tới thực thể Message. Nó kế thừa MongoRepository, một interface trong Spring Data MongoDB, giúp thực hiện các thao tác cơ bản với MongoDB một cách dễ dàng và hiệu quả.

3.8. Xây dựng Service

Sau khi xây dựng các thực thể xong, chúng ta sẽ xây dựng các lớp Service với các thao tác xử lý. Vì lớp Service rất nhiều thao tác nên mã nguồn rất khó quản lý, do vậy, các dự án Java Spring hiện nay đều tạo một lớp trừu tượng để lưu trữ các prototype và làm lớp tương tác, và một lớp kế thừa để thêm các phương thức xử lý.

Lớp trừu tượng *MessageService*:

```
@Service
public interface MessageService {
    ResponseEntity<String> sendMessage(Message messages);
    ResponseEntity<String> deleteMessage(String messageId);
    ResponseEntity<String> editMessage(String messageId, String content);
    ResponseEntity<String> forwardMessage(String messageId, String
conservationId);
    List<Message> getMessagesByConversationId(String conversationId);
}
```

Lớp kế thừa lớp trừu tượng *MessageServiceImpl*:

```
@Service
public class MessageServiceImpl implements MessageService {
    private final MessageRepository messageRepository;
    private final ConversationsRepository conversationsRepository;
    private final UserRepository userRepository;

    public MessageServiceImpl(MessageRepository messageRepository,
        ConversationsRepository
        conversationsRepository,
        UserRepository userRepository) {
        this.messageRepository = messageRepository;
        this.conversationsRepository = conversationsRepository;
        this.userRepository = userRepository;
    }

    @Override
    public ResponseEntity<String> sendMessage(Message messages) {
        Optional<Conversation> conversationOptional =
        conversationsRepository.findById(messages.getConservationId());
        if (conversationOptional.isEmpty())
            return ResponseEntity.badRequest().body("Conversation not
found");

        Optional<User> userOptional =
        userRepository.findById(messages.getUserId());
        if (userOptional.isEmpty())
            return ResponseEntity.badRequest().body("Sender not found");

        if (messages.getContent() == null || messages.getType() == null)
            return ResponseEntity.badRequest().body("Invalid message
content");
    }
}
```



```

        if (messages.getContent().isEmpty())
            return ResponseEntity.badRequest().body("Message content cannot
be empty");

        try {
            messages.setTimestamp(System.currentTimeMillis());
            messageRepository.save(messages);
            return ResponseEntity.ok("Message sent successfully");
        } catch (Exception e) {
            return ResponseEntity.badRequest().body("Bad request" + e);
        }
    }
    ...
}

```

@Service: Annotation này từ Spring, đánh dấu đây là một Service.

Ở đây, ta ví dụ phương thức xử lý là SendMessage. Cú pháp của ngôn ngữ lập trình Java rất chặt chẽ, lớp kế thừa phải đầy đủ các prototype của lớp cha, do vậy, để chương trình chạy được và hợp lệ, lớp con phải kế thừa lớp cha và xử lý đầy đủ.

3.9. Xây dựng RestController

Sau khi xây dựng các lớp xử lý xong, ta sẽ làm một lớp để giao diện có thể tương tác với người dùng.

Trong dự án này, ta sẽ tách riêng phần front-end với phần back-end. Việc tách ra như vậy giúp trang web có thể xử lý linh hoạt hơn, tăng tính bảo mật và giảm bớt việc tải xuống các tài nguyên không cần thiết.

Dưới đây là ví dụ cho lớp MessageController.

```

@RestController
@RequestMapping("/message")
public class MessageController {
    final MessageService messageService;

    public MessageController(MessageService messageService) {
        this.messageService = messageService;
    }

    @PostMapping("/send")
    public ResponseEntity<String> sendMessage(@RequestBody Message messages)
    {
        return messageService.sendMessage(messages);
    }

    @DeleteMapping("/delete")
    public ResponseEntity<String> deleteMessage(@RequestParam String
messageId) {

```

```

        return messageService.deleteMessage(messageId);
    }

    @PostMapping("/edit")
    public ResponseEntity<String> editMessage(@RequestParam String messageId,
String content) {
        return messageService.editMessage(messageId, content);
    }

    @PostMapping("/forward")
    public ResponseEntity<String> forwardMessage(@RequestParam String
messageId, String conservationId) {
        return messageService.forwardMessage(messageId, conservationId);
    }

    @GetMapping("/get")
    public List<Message> getMessagesByConversationId(@RequestParam String
conversationId) {
        return messageService.getMessagesByConversationId(conversationId);
    }
}

```

MessageController là một lớp controller trong Spring Boot, được đánh dấu bằng annotation **@RestController**. Annotation này cho biết lớp này sẽ được sử dụng để xử lý các yêu cầu HTTP với kết quả trả về là JSON.

@RequestMapping("/message") định nghĩa đường dẫn chung cho tất cả các phương thức trong lớp này. Trong trường hợp này, tất cả các yêu cầu đến **/message** sẽ được xử lý bởi lớp MessageController.

MessageService messageService là một dependency được inject vào **MessageController** thông qua constructor. Điều này cho phép ta sử dụng các phương thức của **MessageService** trong **MessageController**.

Các phương thức trong **MessageController** được đánh dấu bằng các annotation như **@PostMapping**, **@DeleteMapping**, **@GetMapping**, tương ứng với các loại yêu cầu HTTP POST, DELETE, GET. Các phương thức này sẽ xử lý các yêu cầu HTTP tương ứng.

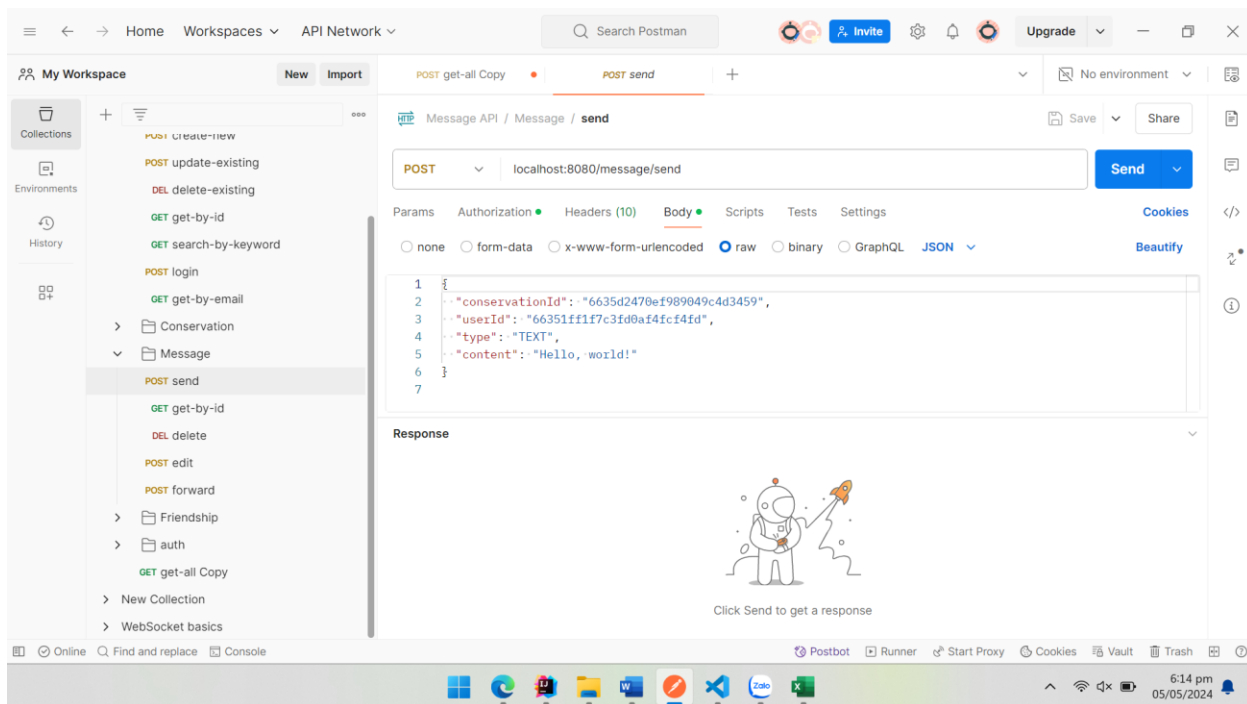
@RequestBody và **@RequestParam** được sử dụng để lấy dữ liệu từ body và tham số của yêu cầu HTTP.

3.10. Kiểm thử với POSTMAN.

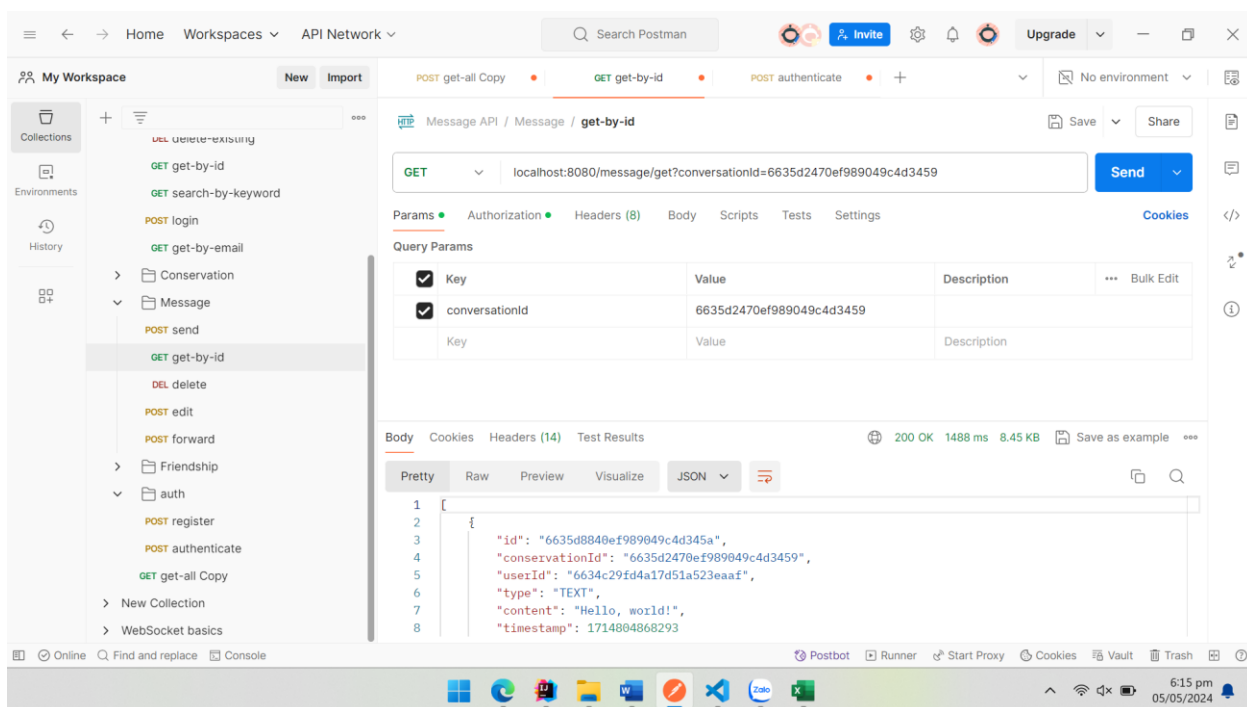
Để kiểm thử RESTful API vừa viết, POSTMAN là công cụ tốt nhất để test.

Ta sẽ tạo một Request trong POSTMAN, tùy theo phương thức của API là GET hay POST ... , request có thể là Body (JSON, XML, ...) hoặc là param.

Dưới đây là một số ví dụ:



Hình 8. Kiểm thử bằng POSTMAN với JSON Body, phương thức POST



Hình 9. Kiểm thử bằng POSTMAN với Params, phương thức GET

Chương 4. CÀI ĐẶT SPRING SECURITY

4.1. Giới thiệu chung về Spring Security

Spring Security là một framework bảo mật mạnh mẽ và linh hoạt cho các ứng dụng Java. Nó cung cấp một loạt các tính năng bảo mật như xác thực, phân quyền, bảo vệ chống tấn công CSRF, và nhiều hơn nữa.

Xác thực: Spring Security hỗ trợ nhiều phương thức xác thực khác nhau, bao gồm xác thực dựa trên form, xác thực HTTP Basic, xác thực dựa trên LDAP, và hơn thế nữa.

Phân quyền: Spring Security cho phép ta kiểm soát quyền truy cập vào các tài nguyên cụ thể dựa trên vai trò của người dùng.

Bảo vệ chống tấn công CSRF: Spring Security cung cấp bảo vệ chống tấn công CSRF (Cross-Site Request Forgery) bằng cách sử dụng một token duy nhất cho mỗi phiên người dùng.

Tích hợp với Spring Boot: Spring Security có thể dễ dàng được tích hợp với Spring Boot, giúp ta cung cấp bảo mật cho ứng dụng của mình một cách nhanh chóng và dễ dàng.

4.2. Các tính năng bảo mật chính được triển khai

4.2.1. Xác thực người dùng

Quá trình xác thực người dùng được tiến hành thông qua việc áp dụng một dịch vụ UserDetailsService được tùy chỉnh. Dịch vụ này chịu trách nhiệm tìm kiếm thông tin chi tiết về người dùng từ cơ sở dữ liệu, dựa trên địa chỉ email được cung cấp. Để đảm bảo tính xác thực của thông tin đăng nhập, một AuthenticationProvider được tùy biến được sử dụng.

4.2.2. Mã hóa mật khẩu

Để đảm bảo tính an toàn của mật khẩu người dùng, BCryptPasswordEncoder được sử dụng để mã hóa mật khẩu trước khi chúng được lưu trữ trong cơ sở dữ liệu. Quy trình này giúp ngăn chặn việc mật khẩu bị lộ hoặc bị đánh cắp trong trường hợp cơ sở dữ liệu bị xâm nhập. Nó kết hợp với PasswordEncoder để kiểm tra và xác nhận tính hợp lệ của thông tin đăng nhập.

```
@Configuration
@RequiredArgsConstructor
public class ApplicationConfig {

    private final UserRepository repository;
```

```

@Bean
public UserDetailsService userDetailsService() {
    return username -> {
        User user = repository.findByEmail(username);
        if (user == null) {
            throw new UsernameNotFoundException("User not found");
        }
        return user;
    };
}

@Bean
public AuthenticationProvider authenticationProvider() {
    DaoAuthenticationProvider authProvider = new
    DaoAuthenticationProvider();
    authProvider.setUserDetailsService(userDetailsService());
    authProvider.setPasswordEncoder(passwordEncoder());
    return authProvider;
}

@Bean
public AuthenticationManager
authenticationManager(AuthenticationConfiguration config) throws Exception {
    return config.getAuthenticationManager();
}

@Bean
public PasswordEncoder passwordEncoder() {
    return new BCryptPasswordEncoder();
}
}

```

- **@Configuration:** Đánh dấu lớp này là một lớp cấu hình Spring.

- **@RequiredArgsConstructor:** Lombok annotation, tạo ra một hàm tạo (constructor) với tất cả các trường cuối cùng (final) và các trường được đánh dấu @NonNull là tham số.

Trong lớp này, ta có một trường:

- UserRepository repository: Một đối tượng UserRepository được sử dụng để truy vấn thông tin người dùng từ cơ sở dữ liệu.

Các phương thức @Bean trong lớp này tạo ra các bean Spring:

- userDetailsService(): Tạo ra một UserDetailsService để tìm kiếm thông tin người dùng từ cơ sở dữ liệu dựa trên email.

- `authenticationProvider()`: Tạo ra một `AuthenticationProvider` để xác thực thông tin đăng nhập. Nó sử dụng `UserDetailsService` để lấy thông tin người dùng và `PasswordEncoder` để kiểm tra mật khẩu.

- `authenticationManager(AuthenticationConfiguration config)`: Tạo ra một `AuthenticationManager` để quản lý quá trình xác thực.

- `passwordEncoder()`: Tạo ra một `PasswordEncoder` để mã hóa mật khẩu. Trong trường hợp này, ta sử dụng `BCryptPasswordEncoder`, một thuật toán mã hóa mật khẩu phổ biến.

```
public PasswordEncoder passwordEncoder() {  
    return new BCryptPasswordEncoder();  
}
```

4.2.3. Xác thực JWT

Sau khi người dùng đăng nhập thành công, một JWT (JSON Web Token) được phát hành. Token này chứa thông tin như tên người dùng và thời gian hết hạn, giúp xác thực các yêu cầu sau này mà không cần nhập lại mật khẩu.

```
@Service  
public class JwtService {  
  
    private static final String SECRET_KEY =  
        "8923d645657c1411cd73677130e8f9eda426b3f*****";  
  
    public String extractUserName(String token) {  
        return extractClaim(token, Claims::getSubject);  
    }  
  
    public <T> T extractClaim(String token, Function<Claims, T>  
claimsResolver) {  
        final Claims claims = extractallClaims(token);  
        return claimsResolver.apply(claims);  
    }  
  
    public String generateToken(UserDetails userDetails) {  
        return generateToken(new HashMap<>(), userDetails);  
    }  
  
    public String generateToken(  
        Map<String, Object> extraClaims,  
        UserDetails userDetails  
    ) {  
        return Jwts  
            .builder()  
            .setClaims(extraClaims)  
            .setSubject(userDetails.getUsername())  
            .setIssuedAt(new Date(System.currentTimeMillis()))
```

```

        .setExpiration(new Date(System.currentTimeMillis() + 1000 *
60 * 60 * 24))
        .signWith(getSignInKey(), SignatureAlgorithm.HS256)
        .compact();
    }

    public boolean isValidToken(String token, UserDetails userDetails) {
        final String username = extractUsername(token);
        return (username.equals(userDetails.getUsername()) &&
!isTokenExpired(token));
    }

    private boolean isTokenExpired(String token) {
        return extractExpiration(token).before(new Date());
    }

    private Date extractExpiration(String token) {
        return extractClaim(token, Claims::getExpiration);
    }

    private Claims extractAllClaims(String token) {
        return Jwts
            .parserBuilder()
            .setSigningKey(getSignInKey())
            .build()
            .parseClaimsJws(token)
            .getBody();
    }

    private Key getSignInKey() {
        byte[] keyBytes = Decoders.BASE64.decode(SECRET_KEY);
        return Keys.hmacShaKeyFor(keyBytes);
    }
}

```

4.2.4. Xác thực và ủy quyền

Hệ thống được thiết lập sao cho mỗi yêu cầu HTTP gửi đến máy chủ đều phải đi qua quá trình xác thực. Bất kỳ yêu cầu nào không đi kèm với JWT hợp lệ sẽ bị từ chối ngay lập tức, đảm bảo rằng chỉ những người dùng đã xác thực thành công mới có quyền truy cập vào các tài nguyên nhạy cảm trên hệ thống. Như vậy, việc này giúp tăng cường bảo mật, bảo vệ thông tin người dùng và ngăn chặn truy cập trái phép.

```

@Configuration
@EnableWebSecurity
@RequiredArgsConstructor
public class SecurityConfiguration {

    private final JwtAuthenticationFilter jwtAuthFilter;
    private final AuthenticationProvider authenticationProvider;
}

```

```

@Bean
public SecurityFilterChain securityFilterChain(HttpSecurity http) throws
Exception {
    http
        .csrf(AbstractHttpConfigurer::disable)
        .authorizeHttpRequests(req ->
            req.requestMatchers("/auth/**", "/ws/**")
                .permitAll()
                .anyRequest()
                .authenticated()
        )
        .sessionManagement(session ->
            session.sessionCreationPolicy(STATELESS))
        .authenticationProvider(authenticationProvider)
        .addFilterBefore(jwtauthFilter,
            UsernamePasswordAuthenticationFilter.class);

    return http.build();
}
}

```

+ @Configuration: Đánh dấu lớp này là một lớp cấu hình Spring.

+ @EnableWebSecurity: Kích hoạt hỗ trợ bảo mật web trong ứng dụng Spring.

+ @RequiredArgsConstructor: Lombok annotation, tạo ra một hàm tạo (constructor) với tất cả các trường cuối cùng (final) và các trường được đánh dấu @NonNull là tham số.

Trong lớp này, ta có hai trường:

+ JwtauthenticationFilter jwtauthFilter: Một bộ lọc xác thực JWT, được sử dụng để kiểm tra xem yêu cầu HTTP có chứa JWT hợp lệ không.

+ AuthenticationProvider authenticationProvider: Một nhà cung cấp xác thực, được sử dụng để xác thực thông tin đăng nhập của người dùng.

Phương thức securityFilterChain(HttpSecurity http) tạo ra một chuỗi bộ lọc bảo mật.

Trong phương thức này, ta cấu hình các yếu tố bảo mật sau:

+ Tắt bảo vệ chống tấn công CSRF.

+ Cho phép tất cả các yêu cầu đến các đường dẫn /auth/** và /ws/**.

+ Yêu cầu xác thực cho tất cả các yêu cầu khác.

+ Đặt chính sách tạo phiên là STATELESS, nghĩa là không sử dụng phiên HTTP.

+ Đặt nhà cung cấp xác thực là authenticationProvider.

Thêm jwtauthFilter vào chuỗi bộ lọc, trước UsernamePasswordAuthenticationFilter.

Phương thức này trả về một SecurityFilterChain, đại diện cho chuỗi bộ lọc bảo mật đã được cấu hình.

4.2.5. Xử lý lỗi

Khi xác thực không thành công, ví dụ như mật khẩu không đúng hoặc token đã hết hạn, Spring Security sẽ phản hồi với thông báo lỗi chi tiết. Điều này giúp người dùng nắm bắt được nguyên nhân gây ra lỗi và tìm cách khắc phục một cách hiệu quả.

```
@Component
@RequiredArgsConstructor
public class JwtauthenticationFilter extends OncePerRequestFilter {

    private final JwtService jwtService;
    private final UserDetailsService userDetailsService;

    @Override
    protected void doFilterInternal(
        @NonNull HttpServletRequest request,
        @NonNull HttpServletResponse response,
        @NonNull FilterChain filterChain
    ) throws ServletException, IOException {
        final String authHeader = request.getHeader("Authorization");
        final String jwt;
        final String userEmail;
        if (authHeader == null || !authHeader.startsWith("Bearer ")) {
            filterChain.doFilter(request, response);
            return;
        }
        jwt = authHeader.substring(7);
        userEmail = jwtService.extractUserName(jwt);
        if (userEmail != null &&
            SecurityContextHolder.getContext().getAuthentication() == null) {
            UserDetails userDetails =
                this.userDetailsService.loadUserByUsername(userEmail);
            if (jwtService.isTokenValid(jwt, userDetails)) {
                UsernamePasswordAuthenticationToken authToken = new
                UsernamePasswordAuthenticationToken(
                    userDetails,
                    null,
                    userDetails.getAuthorities()
                );
                authToken.setDetails(
                    new
                    WebAuthenticationDetailsSource().buildDetails(request)
                );
                SecurityContextHolder.getContext().setAuthentication(authToken);
            }
        }
    }
}
```

```

    }
    filterChain.doFilter(request, response);
}
}

```

- **@Component**: Đánh dấu lớp này là một bean Spring, nghĩa là Spring sẽ quản lý đối tượng của lớp này.
- **@RequiredArgsConstructor**: Lombok annotation, tạo ra một hàm tạo (constructor) với tất cả các trường cuối cùng (final) và các trường được đánh dấu **@NonNull** là tham số.

Trong lớp này, ta có hai trường:

- **JwtService jwtService**: Một đối tượng JwtService được sử dụng để xử lý các hoạt động liên quan đến JWT.
- **UserDetailsService userDetailsService**: Một đối tượng UserDetailsService được sử dụng để tìm kiếm thông tin người dùng từ cơ sở dữ liệu.

Phương thức `doFilterInternal` được ghi đè từ lớp `OncePerRequestFilter`. Phương thức này được gọi mỗi khi một yêu cầu HTTP được gửi đến máy chủ. Trong phương thức này, ta thực hiện các bước sau:

1. Lấy tiêu đề "Authorization" từ yêu cầu.
2. Nếu tiêu đề không tồn tại hoặc không bắt đầu bằng "Bearer ", ta cho phép yêu cầu đi qua chuỗi bộ lọc mà không cần xác thực.
3. Nếu tiêu đề tồn tại và bắt đầu bằng "Bearer ", ta trích xuất JWT từ tiêu đề và trích xuất tên người dùng từ JWT.
4. Nếu tên người dùng tồn tại và người dùng chưa được xác thực, ta tải thông tin người dùng từ cơ sở dữ liệu và kiểm tra xem JWT có hợp lệ không.
5. Nếu JWT hợp lệ, ta tạo một token xác thực mới và đặt nó vào `SecurityContext`, cho phép người dùng được xác thực.
6. Cuối cùng, ta cho phép yêu cầu đi qua chuỗi bộ lọc, cho dù người dùng đã được xác thực hay chưa.

4.2.6. Cơ chế bảo mật CORS:

Mặc định, khi sử dụng Spring Boot RESTful API, các yêu cầu từ nguồn khác sẽ bị từ chối do chính sách Same-Origin Policy (SOP) của trình duyệt. Tuy nhiên, trong một số trường hợp, ta có thể muốn cho phép một số nguồn tin cậy truy cập vào API của mình. Đây là nơi CORS (Cross-Origin Resource Sharing) trở nên quan trọng.

CORS là một cơ chế cho phép nhiều tài nguyên chạy trên một trang web có nguồn gốc khác nhau. Với CORS, máy chủ có thể chỉ định các nguồn khác có thể truy cập vào một số tài nguyên nhất định hoặc tất cả tài nguyên trên máy chủ.

Để cho phép truy cập, ta sẽ tạo một lớp CORS Config để xử lý.

```
package org.vinhveer.message.Config;

import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.CorsRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@Configuration
public class CORSConfig implements WebMvcConfigurer {
    @Override
    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping("/**")
            .allowedOrigins("http://127.0.0.1:5500",
                "http://localhost:3000")
            .allowedMethods("GET", "POST", "PUT", "DELETE");
    }
}
```

- `@Configuration`: Đánh dấu lớp này là một lớp cấu hình Spring.
- `implements WebMvcConfigurer`: Lớp này triển khai giao diện `WebMvcConfigurer` để cấu hình các yếu tố liên quan đến MVC.

Phương thức `addCorsMappings(CorsRegistry registry)` được ghi đè từ giao diện `WebMvcConfigurer`. Phương thức này được sử dụng để cấu hình CORS cho ứng dụng. Trong phương thức này, ta thực hiện các bước sau:

1. `registry.addMapping("/**")`: Cho phép CORS cho tất cả các đường dẫn trong ứng dụng.
2. `.allowedOrigins("http://127.0.0.1:5500", "http://localhost:3000")`: Chỉ cho phép các yêu cầu từ các nguồn đã chỉ định. Trong trường hợp này, chỉ có các yêu cầu từ `http://127.0.0.1:5500` và `http://localhost:3000` mới được phép.

3. `.allowedMethods("GET", "POST", "PUT", "DELETE")`: Chỉ cho phép các phương thức HTTP đã chỉ định. Trong trường hợp này, chỉ có các yêu cầu GET, POST, PUT, và DELETE mới được phép.

Như vậy, với cấu hình này, chỉ có các yêu cầu từ `http://127.0.0.1:5500` và `http://localhost:3000` sử dụng các phương thức GET, POST, PUT, và DELETE mới có thể truy cập vào các tài nguyên của ứng dụng. Các yêu cầu từ nguồn khác hoặc sử dụng các phương thức khác sẽ bị từ chối.

4.3. Xử lý request đăng nhập và đăng ký

Để xử lý đăng nhập, ta sẽ thêm lớp `AuthenticationService`.

```
@Service
@RequiredArgsConstructor
public class AuthenticationService {

    private final UserRepository repository;
    private final PasswordEncoder passwordEncoder;
    private final JwtService jwtService;
    private final AuthenticationManager authenticationManager;

    public AuthenticationResponse register(RegisterRequest request) {
        var user = User.builder()
            .fullName(request.getFullName())
            .gender(request.isGender())
            .dateOfBirth(request.getDateOfBirth())
            .email(request.getEmail())
            .avatar(request.getAvatar())
            .password(passwordEncoder.encode(request.getPassword()))
            .build();
        repository.save(user);
        var jwtToken = jwtService.generateToken(user);
        return AuthenticationResponse.builder()
            .token(jwtToken)
            .build();
    }

    public AuthenticationResponse authenticate(AuthenticationRequest request)
    {
        authenticationManager.authenticate(
            new UsernamePasswordAuthenticationToken(
                request.getEmail(),
                request.getPassword()
            )
        );
        var user = repository.findByEmail(request.getEmail());
        if (user == null) {
            throw new UsernameNotFoundException("User not found");
        }
        var jwtToken = jwtService.generateToken(user);
    }
```

```

        return AuthenticationResponse.builder()
            .token(jwtToken)
            .build();
    }
}

```

+ **@Service**: Đây là một annotation của Spring, biểu thị rằng lớp này là một lớp dịch vụ. Spring sẽ quản lý các instance của lớp này như là một bean trong IoC container.

+ **@RequiredArgsConstructor**: Đây là một annotation của Lombok, sẽ tự động tạo ra một constructor với tất cả các trường final.

+ **register(RegisterRequest request)**: Phương thức này được sử dụng để đăng ký một người dùng mới. Nó nhận vào một đối tượng RegisterRequest chứa thông tin của người dùng, sau đó mã hóa mật khẩu và lưu người dùng vào cơ sở dữ liệu. Cuối cùng, nó tạo một token JWT cho người dùng và trả về.

+ **authenticate(AuthenticationRequest request)**: Phương thức này được sử dụng để xác thực người dùng. Nó nhận vào một đối tượng AuthenticationRequest chứa email và mật khẩu, sau đó sử dụng AuthenticationManager để xác thực. Nếu xác thực thành công, nó sẽ tạo một token JWT và trả về.

Để người dùng có thể tương tác được, ta sẽ thêm một lớp Controller:

```

@RestController
@RequestMapping("/auth")
@RequiredArgsConstructor
public class AuthenticationController {

    private final AuthenticationService service;

    @PostMapping("/register")
    public ResponseEntity<AuthenticationResponse> register(
        @RequestBody RegisterRequest request
    ) {
        return ResponseEntity.ok(service.register(request));
    }

    @PostMapping("/authenticate")
    public ResponseEntity<AuthenticationResponse> authenticate(
        @RequestBody AuthenticationRequest request
    ) {
        return ResponseEntity.ok(service.authenticate(request));
    }
}

```

+ **@RestController**: Annotation này biểu thị rằng lớp này là một REST Controller. Các phương thức trong lớp này sẽ trả về dữ liệu dưới dạng JSON hoặc XML tới client.

+ **@RequestMapping("/auth")**: Annotation này định nghĩa đường dẫn chung cho tất cả các phương thức trong lớp này. Trong trường hợp này, tất cả các phương thức sẽ có đường dẫn bắt đầu bằng "/auth".

+ **@RequiredArgsConstructor**: Đây là một annotation của Lombok, sẽ tự động tạo ra một constructor với tất cả các trường final.

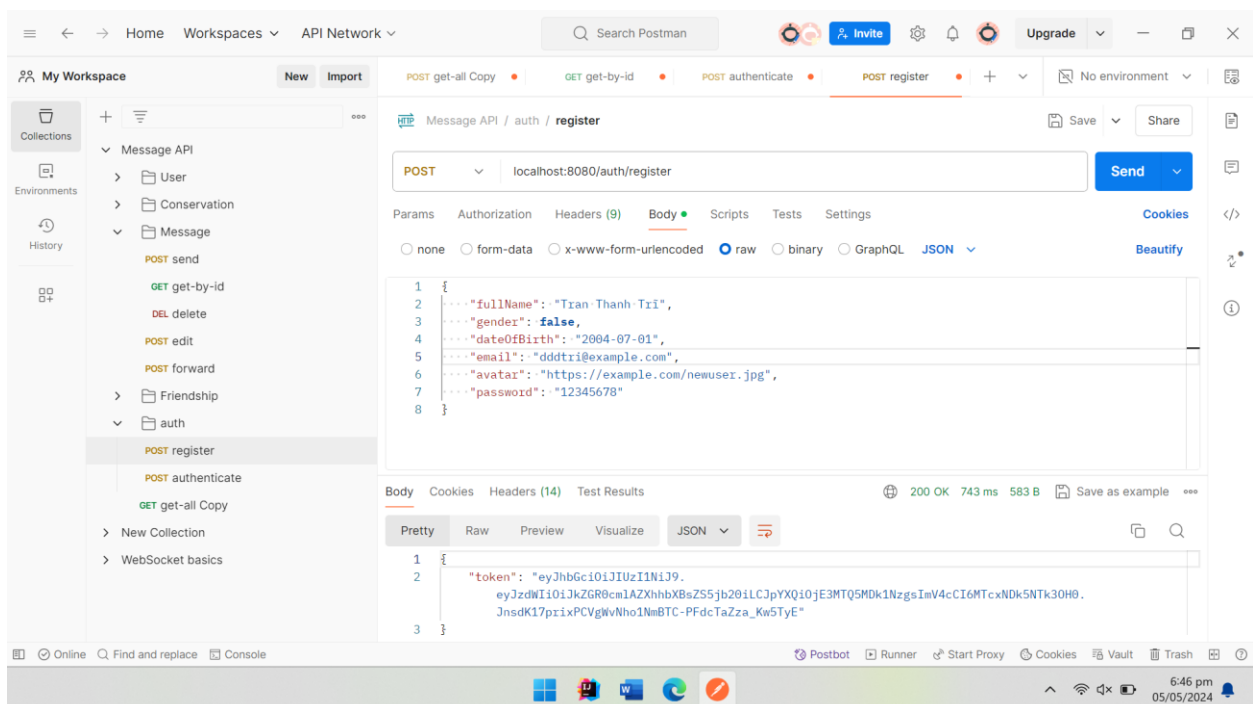
+ **register(@RequestBody RegisterRequest request)**: Phương thức này xử lý yêu cầu POST tới "/auth/register". Nó nhận vào một đối tượng RegisterRequest từ body của yêu cầu, sau đó gọi phương thức register của AuthenticationService để đăng ký người dùng mới.

+ **authenticate(@RequestBody AuthenticationRequest request)**: Phương thức này xử lý yêu cầu POST tới "/auth/authenticate". Nó nhận vào một đối tượng AuthenticationRequest từ body của yêu cầu, sau đó gọi phương thức authenticate của AuthenticationService để xác thực người dùng.

4.4. Kiểm thử đăng nhập và đăng ký

Để kiểm thử, ta sẽ sử dụng công cụ POSTMAN.

Dưới đây là ví dụ:



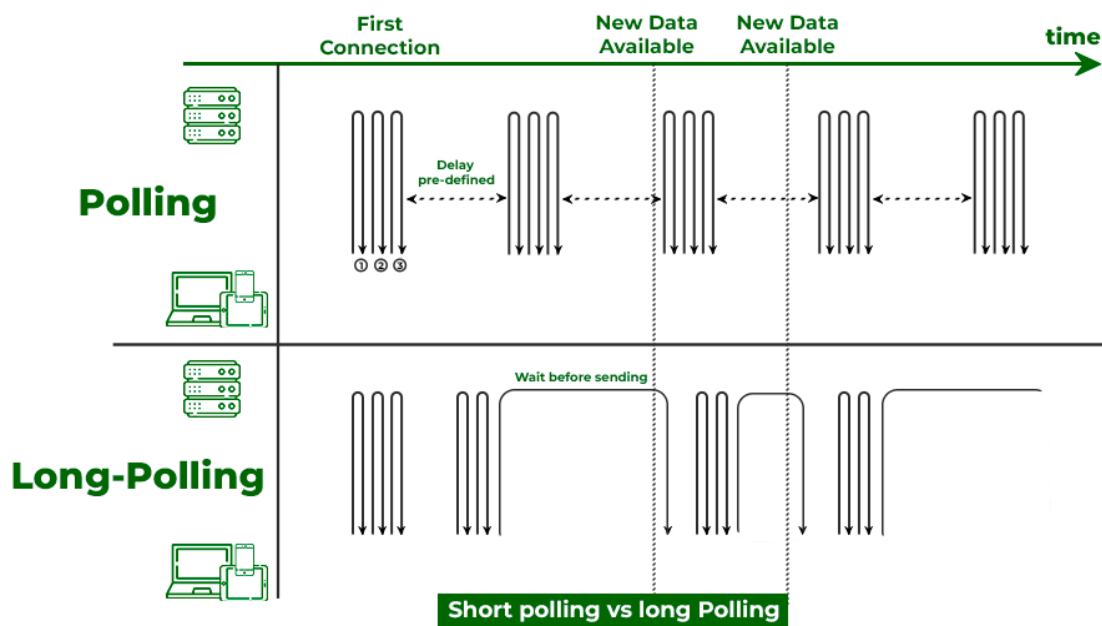
Hình 10. Kiểm thử API register với POSTMAN

Chương 5. CÀI ĐẶT WEBSOCKET

5.1. Khái niệm cơ bản về WebSocket

WebSocket là một giao thức cho phép thiết lập kết nối TCP liên tục giữa máy chủ và máy khách để họ có thể trao đổi dữ liệu bất cứ lúc nào.

Trước khi có WebSocket, để xây dựng các ứng dụng trực tuyến như nhắn tin ..., người ta thường sử dụng hai cơ chế là Polling và về sau là Long - Polling



Hình 12. Cơ chế của Polling và Long Polling

Polling (Short Polling):

- + Trong Short Polling, máy khách gửi yêu cầu đến máy chủ theo khoảng thời gian cố định.

- + Máy chủ sẽ trả lời ngay lập tức, nếu có dữ liệu sẵn có thì trả về dữ liệu, nếu không thì trả về một phản hồi trống.

- + Sau khi nhận được phản hồi từ máy chủ, máy khách sẽ đợi một khoảng thời gian cố định và sau đó gửi yêu cầu tiếp theo.

- + Short Polling đơn giản và không tiêu tốn nhiều tài nguyên máy chủ, nhưng thông báo sự kiện không tức thì và có thể có độ trễ.

Long Polling:

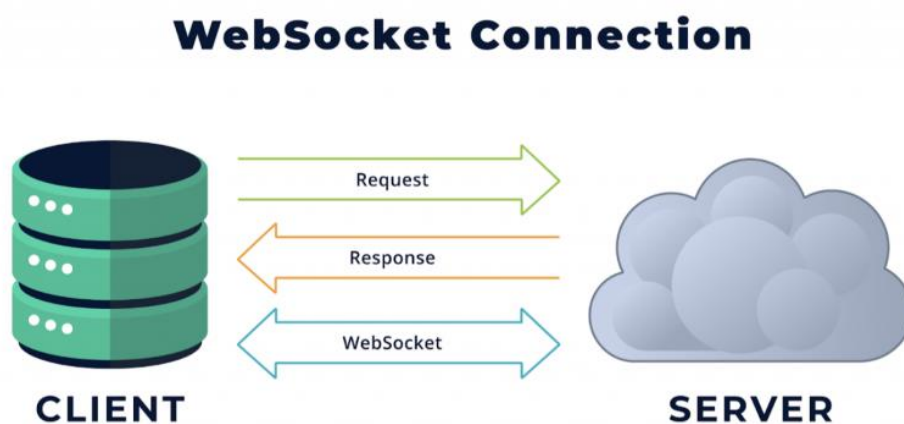
- + Trong Long Polling, máy khách cũng gửi yêu cầu đến máy chủ.

+ Tuy nhiên, nếu không có dữ liệu sẵn có, thay vì trả về một phản hồi trống, máy chủ sẽ giữ yêu cầu và chờ đợi cho đến khi có dữ liệu mới¹²⁴⁵.

+ Khi có dữ liệu mới, máy chủ sẽ trả lời yêu cầu với dữ liệu mới này¹²⁴⁵.

+ Long Polling phức tạp hơn và có thể tiêu tốn nhiều tài nguyên máy chủ hơn, nhưng nó cung cấp thông báo sự kiện gần như tức thì, do đó mang lại trải nghiệm thời gian thực tốt hơn.

Và để giải quyết nhược điểm của các cơ chế này, WebSocket ra đời, cơ chế của WebSocket có thể mô tả qua hình sau:



Hình 13. Cơ chế của WebSocket

WebSocket là một công nghệ cho phép truyền tải dữ liệu hai chiều giữa máy chủ và trình duyệt web hoặc ứng dụng web một cách liên tục và hiệu quả. Thay vì cách thông thường là máy khách gửi yêu cầu và máy chủ trả lời (request-response), WebSocket cho phép cả hai bên giao tiếp cùng một lúc, tạo ra một kết nối liên tục.

Khi một kết nối WebSocket được thiết lập, nó bắt đầu bằng một handshake thông qua HTTP. Sau khi handshake hoàn tất, kết nối được chuyển sang một kết nối TCP liên tục, giúp tránh các overhead của việc thiết lập kết nối mới mỗi khi cần truyền tải dữ liệu. Điều này cho phép truyền tải dữ liệu theo thời gian thực một cách nhanh chóng và hiệu quả hơn.

5.2. Cài đặt WebSocket vào dự án Spring Boot

Trước khi cài đặt, ta cần có dependency tên là WebSocket (Đã được cài đặt ở Chương 3).

Đầu tiên, ta tạo lớp **WebSocketConfig** để cấu hình WebSocket cho dự án.

```

@Configuration
@EnableWebSocket
public class WebSocketConfig implements WebSocketConfigurer {
    private final MessageService messageService;
    private final ObjectMapper objectMapper;

    public WebSocketConfig(MessageService messageService, ObjectMapper
objectMapper) {
        this.messageService = messageService;
        this.objectMapper = objectMapper;
    }

    @Override
    public void registerWebSocketHandlers(WebSocketHandlerRegistry registry)
{
        registry.addHandler(new MessageWebSocketHandler(messageService,
objectMapper), "/ws").setAllowedOrigins("*");
    }
}

```

- + **@Configuration**: Đánh dấu lớp này là một lớp cấu hình Spring.
- + **@EnableWebSocket**: Kích hoạt hỗ trợ WebSocket trong ứng dụng Spring.
- + **implements WebSocketConfigurer**: Lớp này kế thừa từ lớp **WebSocketConfigurer** (Một lớp nằm trong thư viện WebSocket của Spring) để cấu hình các yếu tố liên quan đến WebSocket.

Trong lớp này, ta có hai trường:

- + **MessageService messageService**: Một đối tượng **MessageService** được sử dụng để xử lý các tin nhắn.
- + **ObjectMapper objectMapper**: Một đối tượng **ObjectMapper** từ thư viện Jackson, được sử dụng để chuyển đổi giữa các đối tượng Java và JSON.

Phương thức **registerWebSocketHandlers** được ghi đè từ lớp **WebSocketConfigurer**. Phương thức này được sử dụng để đăng ký các trình xử lý WebSocket với Spring. Trong phương thức này, ta tạo một đối tượng **MessageWebSocketHandler** mới với **messageService** và **objectMapper**, và đăng ký nó với đường dẫn **/ws**. Phương thức **setAllowedOrigins("*")** cho phép các yêu cầu từ bất kỳ nguồn gốc nào.

Tiếp theo, ta tạo một lớp là **MessageWebSocketHandler** kế thừa từ lớp **TextWebSocketHandler** (Một lớp nằm trong thư viện WebSocket của Spring)

```

@Component
public class MessageWebSocketHandler extends TextWebSocketHandler {
    private final MessageService messageService;
    private final ObjectMapper objectMapper;
    private final Map<String, WebSocketSession> sessions = new
ConcurrentHashMap<>();

    public MessageWebSocketHandler(MessageService messageService,
ObjectMapper objectMapper) {
        this.messageService = messageService;
        this.objectMapper = objectMapper;
    }

    @Override
    public void afterConnectionEstablished(WebSocketSession session) throws
Exception {
        System.out.println("WebSocket connection established: " +
session.getId());
        sessions.put(session.getId(), session);
    }

    @Override
    protected void handleTextMessage(WebSocketSession session, TextMessage
message) throws Exception {
        Message msg = objectMapper.readValue(message.getPayload(),
Message.class);
        messageService.sendMessage(msg);
        TextMessage response = new
TextMessage(objectMapper.writeValueAsString(msg));
        for (WebSocketSession webSocketSession : sessions.values()) {
            if (!webSocketSession.getId().equals(session.getId()))
                webSocketSession.sendMessage(response);
        }
    }

    @Override
    public void handleTransportError(WebSocketSession session, Throwable
exception) throws Exception {
        super.handleTransportError(session, exception);
    }

    @Override
    public void afterConnectionClosed(WebSocketSession session, CloseStatus
status) throws Exception {
        sessions.remove(session.getId());
    }
}

```

+ **@Component**: Đánh dấu lớp này là một bean Spring, nghĩa là Spring sẽ quản lý đối tượng của lớp này.

+ **extends TextWebSocketHandler**: Lớp này kế thừa từ **TextWebSocketHandler**, một lớp trừu tượng cung cấp các phương thức để xử lý các tin nhắn WebSocket dạng văn bản.

Trong lớp này, ta có ba trường:

+ **MessageService messageService**: Một đối tượng **MessageService** được sử dụng để xử lý các tin nhắn.

+ **ObjectMapper objectMapper**: Một đối tượng **ObjectMapper** từ thư viện Jackson, được sử dụng để chuyển đổi giữa các đối tượng Java và JSON.

+ **Map<String, WebSocketSession> sessions**: Một Map lưu trữ các phiên WebSocket, với ID của phiên làm key.

+ Phương thức **afterConnectionEstablished** được gọi sau khi một kết nối WebSocket được thiết lập. Trong phương thức này, ta thêm phiên mới vào Map sessions.

+ Phương thức **handleTextMessage** được gọi khi một tin nhắn văn bản được nhận từ máy khách. Trong phương thức này, ta chuyển đổi tin nhắn từ JSON thành đối tượng **Message**, gửi tin nhắn đến **MessageService**, và sau đó gửi tin nhắn đến tất cả các phiên khác.

+ Phương thức **handleTransportError** được gọi khi có lỗi xảy ra trong quá trình truyền dữ liệu. Trong phương thức này, ta gọi phương thức **handleTransportError** của lớp cha.

+ Phương thức **afterConnectionClosed** được gọi sau khi một kết nối WebSocket được đóng. Trong phương thức này, ta xóa phiên đã đóng khỏi Map sessions.

5.3. Config Spring Sercurity cho WebSocket.

Để thuận tiện cho việc testing, ta sẽ allow tất cả kết nối có đuôi là **/ws/**** trong lớp **SercurityConfigure**.

```
authorizeHttpRequests(req ->
    req.requestMatchers("/auth/**", "/ws/**")
        .permitAll()
        .anyRequest()
        .authenticated()
)
```

Tuy nhiên, khi trong dự án thực tế, ta nên tạo lớp config Spring Sercurity riêng, vì nếu làm như vậy thì ai cũng có thể lấy data từ WebSocket được, dễ dàng bị hacker xâm nhập lấy đi tin nhắn.

Chương 6. KẾT NỐI RESTful API VỚI GIAO DIỆN WEB BẰNG JAVASCRIPT

6.1. Vẽ giao diện

Để viết giao diện web, ta sử dụng HTML, CSS và JavaScript. Sau khi viết, ta có được giao diện như các hình ở Chương 8 (Kết quả).

6.2. Đăng nhập

Để đăng nhập, ta sẽ truy vấn đến API, nếu đăng nhập thành công, API sẽ trả về token và token này sẽ lưu vào trong localStorage để cấp quyền truy cập cho các trang khác.

```
document.getElementById('loginForm').addEventListener('submit',
function(event) {
    // Ngăn chặn hành vi mặc định của form
    event.preventDefault();

    // Lấy dữ liệu từ form
    const username = document.getElementById('username').value;
    const password = document.getElementById('password').value;

    // Tạo đối tượng request
    const requestData = {
        email: username,
        password: password
    };

    // Thực hiện fetch API
    fetch('http://localhost:8080/auth/authenticate', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify(requestData)
    })
    .then(response => {
        if (!response.ok || response.status !== 200) {
            throw new Error('<div class="alert alert-danger"
role="alert"><strong>Thông báo!</strong> Email hoặc mật khẩu không
đúng.</div>');
        }
        return response.json();
    })
    .then(data => {
        if (data && data.token) {
            // Lưu token vào Storage
```

```

        localStorage.setItem('token', data.token);

        // Chuyển hướng trang
        window.location.href = 'app/index.html';
    } else {
        // Hiển thị thông báo lỗi
        document.querySelector('.notification').innerHTML = '<div
class="alert alert-danger" role="alert"><strong>Thông báo!</strong> Email hoặc
mật khẩu không đúng.</div>';
    }
})
.catch(error => {
    console.error('Error:', error);
    document.querySelector('.notification').innerHTML = error.message;

    // Xóa token khỏi Local Storage
    localStorage.removeItem('token');
});
});

```

var messageInput = document.getElementById("i"); và var sendButton = document.querySelector(".input button"); Lấy phần tử input để nhập tin nhắn và nút gửi từ DOM.

var socket = new WebSocket('ws://localhost:8080/ws'); Khởi tạo một kết nối WebSocket đến máy chủ tại 'ws://localhost:8080/ws'.

sendButton.addEventListener("click", async function (event) {...}); Thêm một sự kiện click vào nút gửi. Khi nút được nhấn, hàm bên trong sẽ được thực thi.

event.preventDefault(); Ngăn chặn hành vi mặc định của form (gửi POST request). Kiểm tra xem người dùng đã chọn một cuộc trò chuyện để gửi tin nhắn chưa. Nếu không, hiển thị thông báo yêu cầu chọn một cuộc trò chuyện.

Lấy nội dung tin nhắn từ input. Nếu input trống, hiển thị thông báo yêu cầu nhập nội dung tin nhắn.

Lấy id của người nhận tin nhắn từ cuộc trò chuyện đang được chọn.

const conservation = await getConservation(friendId); Gọi hàm getConservation để lấy thông tin về cuộc trò chuyện với người bạn đang được chọn.

Tạo một đối tượng tin nhắn mới với id cuộc trò chuyện, id người dùng hiện tại, loại tin nhắn là "TEXT", và nội dung tin nhắn.

socket.send(JSON.stringify(newMessage));: Gửi tin nhắn qua WebSocket dưới dạng chuỗi JSON.

displaySentMessage(newMessage);: Hiển thị tin nhắn đã gửi lên giao diện người dùng.

messageInput.value = "";: Xóa nội dung trong input sau khi tin nhắn đã được gửi.

6.3. Truy vấn đến API.

Để truy vấn đến API, ta sử dụng JavaScript để thực hiện.

Ta sẽ dùng một hàm chung để GET API.

```
function fetchApi(url, method = "GET", body = null) {
  let token = localStorage.getItem("token");

  let headers = new Headers();
  headers.append("Authorization", "Bearer " + token);

  let options = {
    method: method,
    headers: headers,
  };

  if (body) {
    options.body = JSON.stringify(body);
    headers.append("Content-Type", "application/json");
  }

  let request = new Request(url, options);

  return fetch(request)
    .then((response) => response.json())
    .catch((error) => console.error("Error:", error));
}
```

Trong đó:

url: Đây là URL của API mà bạn muốn gửi yêu cầu.

method: Phương thức HTTP mà bạn muốn sử dụng (GET, POST, PUT, DELETE, v.v.). Mặc định là “GET”.

body: Đối tượng chứa dữ liệu mà bạn muốn gửi đến API (chỉ áp dụng cho các yêu cầu POST và PUT). Mặc định là null.

token: Một token được lưu trữ trong localStorage, thường được sử dụng để xác thực người dùng.

headers: Một đối tượng Headers mới được tạo và một header “Authorization” được thêm vào với giá trị là “Bearer ” cộng với token.

options: Một đối tượng chứa phương thức và headers cho yêu cầu. Nếu có body, nó sẽ được chuyển đổi thành chuỗi JSON và thêm vào options, và một header “Content-Type” với giá trị “application/json” cũng sẽ được thêm vào headers.

request: Một đối tượng Request mới được tạo với url và options.

Cuối cùng, hàm fetch được gọi với request. Nó trả về một Promise, và khi Promise này được giải quyết, nó sẽ trả về phản hồi từ API dưới dạng JSON. Nếu có lỗi xảy ra, nó sẽ được ghi lại trong console.

Các hàm tiếp theo đều sử dụng API này để truy vấn và trả kết quả qua DOM đến người dùng.

6.4. Xử lý nhắn tin trực tiếp với WebSocket.

Để xử lý WebSocket, ta sử dụng hàm sau để thực hiện.

```
/ SEND MESSAGE
var messageInput = document.getElementById("i");
var sendButton = document.querySelector(".input button");

// Khởi tạo kết nối WebSocket với token xác thực trong header
var socket = new WebSocket('ws://localhost:8080/ws');

sendButton.addEventListener("click", async function (event) {
    event.preventDefault(); // Ngăn form gửi đi

    // Kiểm tra xem một cuộc trò chuyện có được chọn không
    var activeUserChat = document.querySelector(".userChat.active");
    if (!activeUserChat) {
        alert("Vui lòng chọn một cuộc trò chuyện để gửi tin nhắn.");
        return;
    }

    // Lấy nội dung tin nhắn từ input
    var message = messageInput.value;
    if (message === "") {
        alert("Vui lòng nhập nội dung tin nhắn.");
        return;
    }
}
```



```

// Lấy id của người nhận tin nhắn
var friendId = activeUserChat.dataset.friendId;

const conversation = await getConversation(friendId);

// Tạo đối tượng tin nhắn mới
var newMessage = {
  conversationId: conversation.id,
  userId: currentUserId,
  type: "TEXT",
  content: message,
};

// Gửi tin nhắn qua WebSocket
socket.send(JSON.stringify(newMessage));

// Hiển thị tin nhắn đã gửi
displaySentMessage(newMessage);

messageInput.value = "";
});

```

`document.getElementById('loginForm').addEventListener('submit', function(event) {...});` Thêm một sự kiện submit vào form đăng nhập. Khi form được submit, hàm bên trong sẽ được thực thi.

`event.preventDefault();` Ngăn chặn hành vi mặc định của form (gửi POST request).

Lấy dữ liệu từ form: username và password.

Tạo một đối tượng request với email và password từ form.

Thực hiện fetch API đến endpoint ‘`http://localhost:8080/auth/authenticate`’ với phương thức ‘POST’, header ‘Content-Type: application/json’, và body là chuỗi JSON của đối tượng request.

Kiểm tra response: nếu response không ok hoặc status code không phải 200, ném ra lỗi với thông báo ‘Email hoặc mật khẩu không đúng.’.

Nếu response ok, chuyển response thành JSON.

Kiểm tra data: nếu có token, lưu token vào Local Storage và chuyển hướng trang đến ‘`app/index.html`’. Nếu không, hiển thị thông báo lỗi ‘Email hoặc mật khẩu không đúng.’.

Xử lý lỗi: nếu có lỗi, hiển thị thông báo lỗi và xóa token khỏi Local Storage.

Chương 7. TÍCH HỢP TÍNH NĂNG VIDEOCALL VÀO DỰ ÁN

7.1. Giới thiệu

Mục đích của tính năng: Việc tích hợp tính năng call video vào ứng dụng nhắn tin nhằm mục đích tăng cường sự kết nối giữa người dùng, cho phép họ giao tiếp trực tiếp và hiệu quả hơn trong mọi hoàn cảnh. Tính năng này không chỉ cải thiện trải nghiệm người dùng mà còn thúc đẩy khả năng tiếp cận và linh hoạt của ứng dụng.

Công nghệ sử dụng: Chúng tôi đã lựa chọn ZegoCloud để cung cấp nền tảng cho tính năng call video do khả năng mở rộng, bảo mật cao và độ trễ thấp trong truyền dẫn video. ZegoCloud hỗ trợ đảm bảo chất lượng truyền thông tốt nhất và độ ổn định cần thiết để mang đến những cuộc gọi video liền mạch và an toàn cho người dùng.

7.2. Thiết kế và Cài đặt

7.2.1. Kiến trúc tổng quan

Ứng dụng nhắn tin của chúng tôi được thiết kế để tích hợp mượt mà các tính năng nhắn tin cơ bản cùng với tính năng call video phức tạp hơn. Cấu trúc của ứng dụng bao gồm một front-end, được xây dựng bằng JavaScript và frameworks phổ biến như React hay Vue.js, và một back-end, hỗ trợ cơ sở dữ liệu và các API liên quan đến việc truyền thông. Tính năng call video được tích hợp thông qua ZegoCloud, một nền tảng mạnh mẽ cung cấp khả năng giao tiếp video trực tuyến. Việc tích hợp được thực hiện thông qua các API của ZegoCloud, cho phép chúng tôi tùy chỉnh và kiểm soát hoàn toàn các cuộc gọi, từ khởi tạo đến quản lý trạng thái cuộc gọi.

7.2.2. Chi tiết kỹ thuật

Ứng dụng sử dụng JavaScript cho cả client-side và server-side logic. Trong đó, hàm `initializeCall()` và `getUrlParams()` đóng vai trò quan trọng trong việc khởi tạo và quản lý cuộc gọi video.

+ **`getUrlParams()`:** Hàm này được sử dụng để lấy các tham số từ URL, điều này rất hữu ích trong việc xác định thông tin của phòng chat video mà người dùng sẽ tham gia. Phương thức này tách chuỗi URL và trả về các giá trị tham số cần thiết, như `roomId`, từ đó hỗ trợ việc liên kết phòng.

```
function getUrlParams(url = window.location.href) {
  let urlStr = url.split("?")[1];
  return new URLSearchParams(urlStr);
}
```

initializeCall(): Hàm này bắt đầu quá trình thiết lập cuộc gọi video. Nó xác định người dùng hiện tại và phòng chat thông qua các ID tương ứng, sau đó sử dụng các thông tin này để tạo token từ ZegoCloud, đảm bảo mỗi cuộc gọi được xác thực và quản lý an toàn. Cuộc gọi sau đó được khởi tạo thông qua API của ZegoCloud, cho phép người dùng tham gia phòng.

```
async function initializeCall() {
  const kitToken = ZegoUIKitPrebuilt.generateKitTokenForTest(
    appID,
    serverSecret,
    roomID,
    userID,
    userName
  );

  const zp = ZegoUIKitPrebuilt.create(kitToken);

  zp.joinRoom({
    container: document.querySelector("#root"),
    sharedLinks: [
      {
        url:
          window.location.protocol +
          "://" +
          window.location.host +
          window.location.pathname +
          "?roomID=" +
          roomID,
      },
    ],
    onLeaveRoom: () => {
      root.style.display = "none";
    },
    onReturnToHomeScreenClicked: () => {
      root.style.display = "none";
    },
    scenario: {
      mode: ZegoUIKitPrebuilt.OneONoneCall,
    },
  });
}
```

7.2.3. Quản lý trạng thái

Quản lý trạng thái người dùng trong quá trình cuộc gọi là một phần thiết yếu của ứng dụng. `initializeCall()` không chỉ khởi tạo cuộc gọi mà còn theo dõi trạng thái của người dùng trong cuộc gọi. Khi người dùng tham gia hoặc rời phòng, ứng dụng cập nhật trạng thái này để phản ánh thay đổi trong giao diện người dùng. Việc lựa chọn người dùng để thực hiện cuộc gọi dựa trên thông tin trạng thái hoạt động, đảm bảo rằng cuộc gọi chỉ được thực hiện khi người dùng sẵn sàng và có mặt.

7.3. Tính năng và Chức năng

7.3.1. Giao diện người dùng

Giao diện người dùng của tính năng call video được thiết kế để đơn giản và trực quan, đảm bảo người dùng có thể dễ dàng thực hiện và điều khiển cuộc gọi. Cửa sổ video hiển thị hình ảnh của cả hai bên và bao gồm các nút điều khiển cơ bản như tắt tiếng, tắt video, và chuyển đổi camera. Giao diện tối ưu hóa cho cả di động và máy tính để bàn, đảm bảo trải nghiệm người dùng nhất quán trên mọi thiết bị.

7.3.2. Bảo mật và Quyền riêng tư

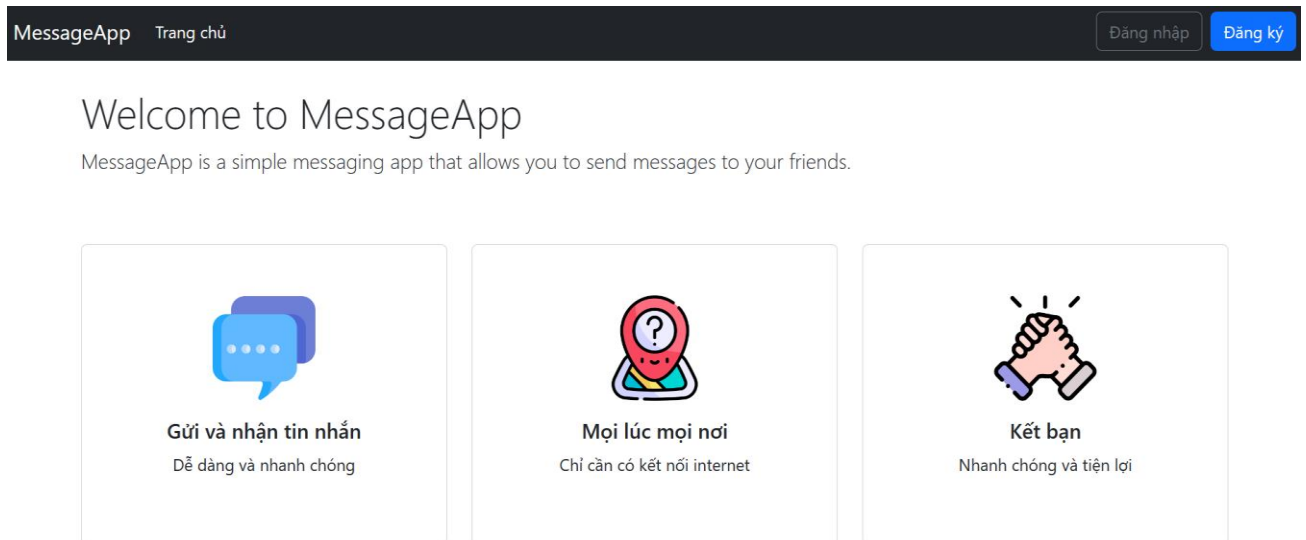
Chúng tôi áp dụng mã hóa end-to-end cho tất cả cuộc gọi video, đảm bảo rằng chỉ các bên tham gia mới có thể truy cập nội dung cuộc gọi. Các biện pháp bảo mật khác bao gồm xác thực đa yếu tố và quản lý quyền truy cập nhằm bảo vệ dữ liệu người dùng khỏi các mối đe dọa bảo mật. Điều này giúp tạo ra một môi trường an toàn cho người dùng khi giao tiếp trực tuyến.

Chương 8. KẾT QUẢ

Sau khi triển khai các bước trên, dự án đã thành công cho ra các kết quả dưới đây:

8.1. Trang chủ và đăng nhập

Đầu tiên khi vào trang web, người dùng sẽ vào trang chủ, có giao diện như hình.



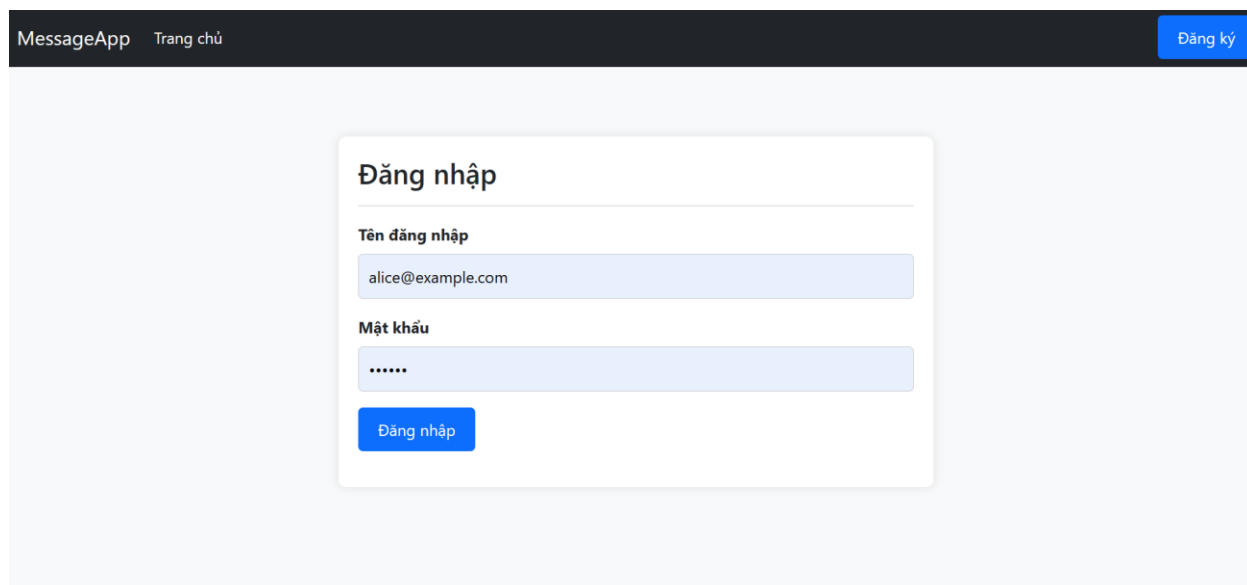
Hình 14. Giao diện trang chủ

Khi click vào đăng ký, sẽ ra giao diện như hình. Người dùng có thể đăng ký dễ dàng qua email (Vì nguồn lực có hạn nên nhóm không thể làm xác thực OTP cho email được)

The screenshot shows the registration form of MessageApp. At the top, there is a dark navigation bar with 'MessageApp' and 'Trang chủ' on the left, and a 'Đăng nhập' button on the right. The main content area is light gray. In the center, there is a white registration form with a title 'Đăng ký'. The form contains four input fields: 'Tên đăng nhập' with the value 'Chau@example.com', 'Email' with the value 'example@example.com', 'Mật khẩu' with a masked password '.....', and 'Nhập lại mật khẩu' with the value '@PassWoRD'. At the bottom of the form, there are two blue buttons: 'Đăng ký' and 'Quay lại'.

Hình 15. Form đăng ký

Trong trường hợp đã có tài khoản, người dùng sẽ click vào Đăng nhập và đăng nhập vào tài khoản của mình.



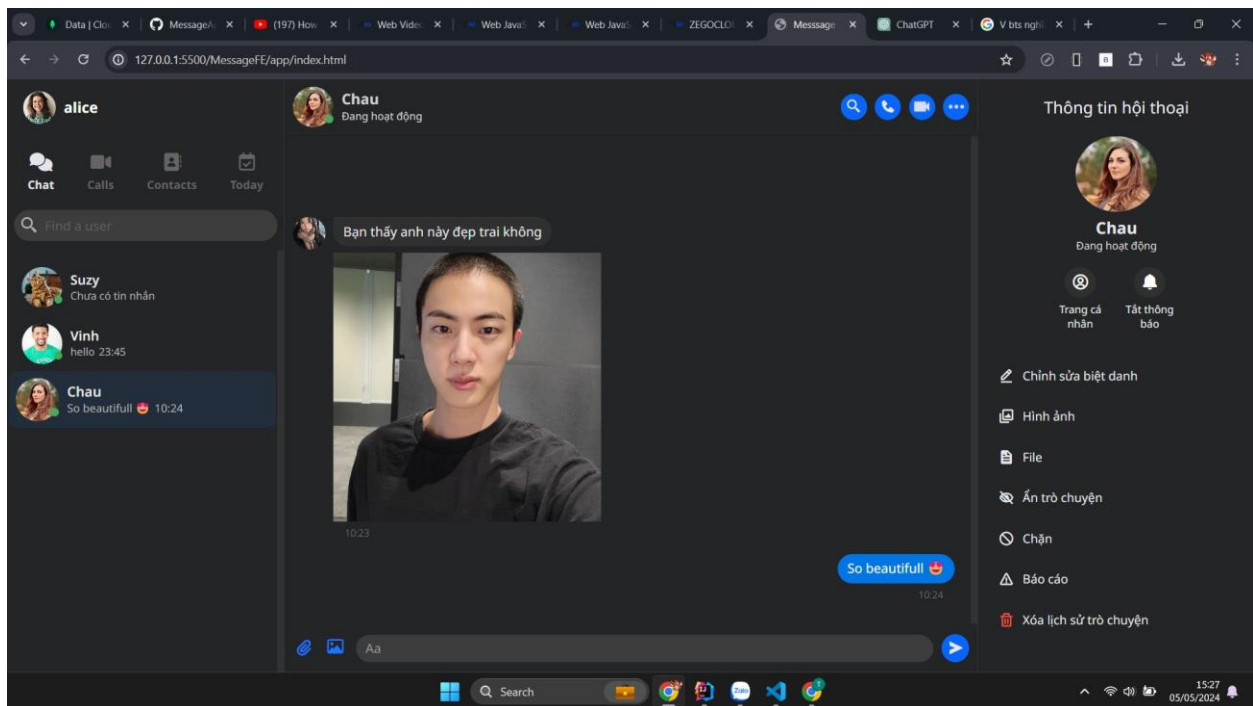
Hình 16. Form đăng nhập

8.2. Tính nhắn nhắn tin với nhiều định dạng

Tiếp theo, sau khi đăng nhập thành công là giao diện tương tác giữa hai người dùng. Ở giao diện này, người dùng có thể tìm kiếm bạn bè, kết bạn, trò chuyện theo thời gian thực như nhắn tin và chia sẻ hình ảnh, video và tệp tin.

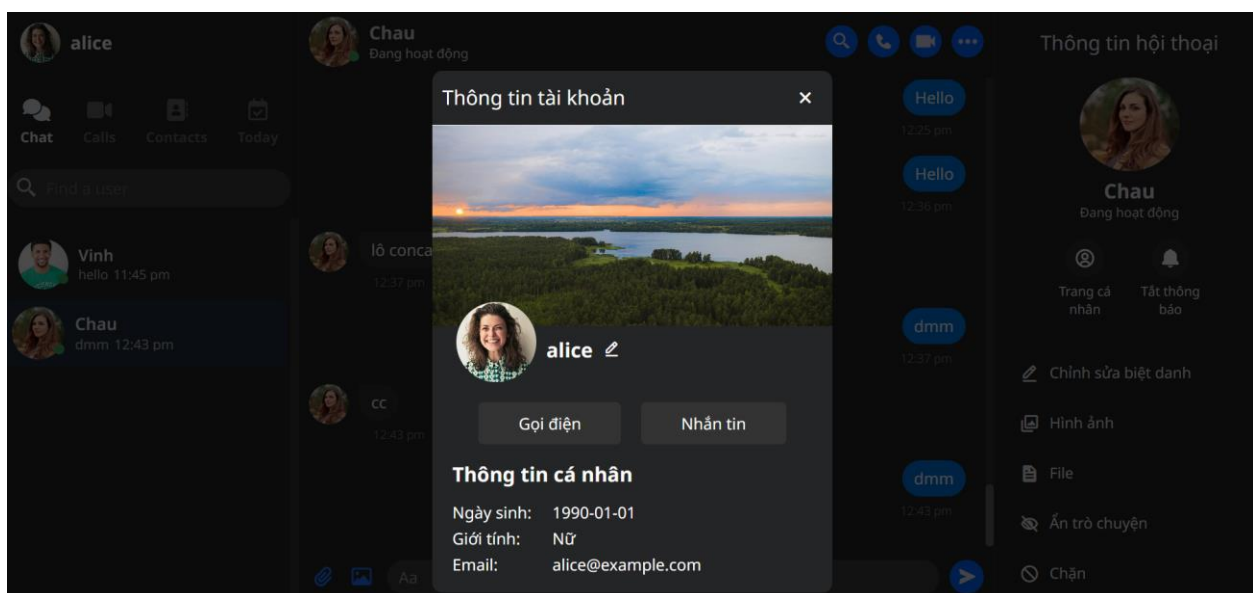


Hình 17. Tính năng nhắn tin



Hình 18. Tính năng gửi file hình ảnh, video, tệp tin,...

Ta có thể dễ dàng xem các thông tin cá nhân của bản thân bằng cách click vào “Trang cá nhân”



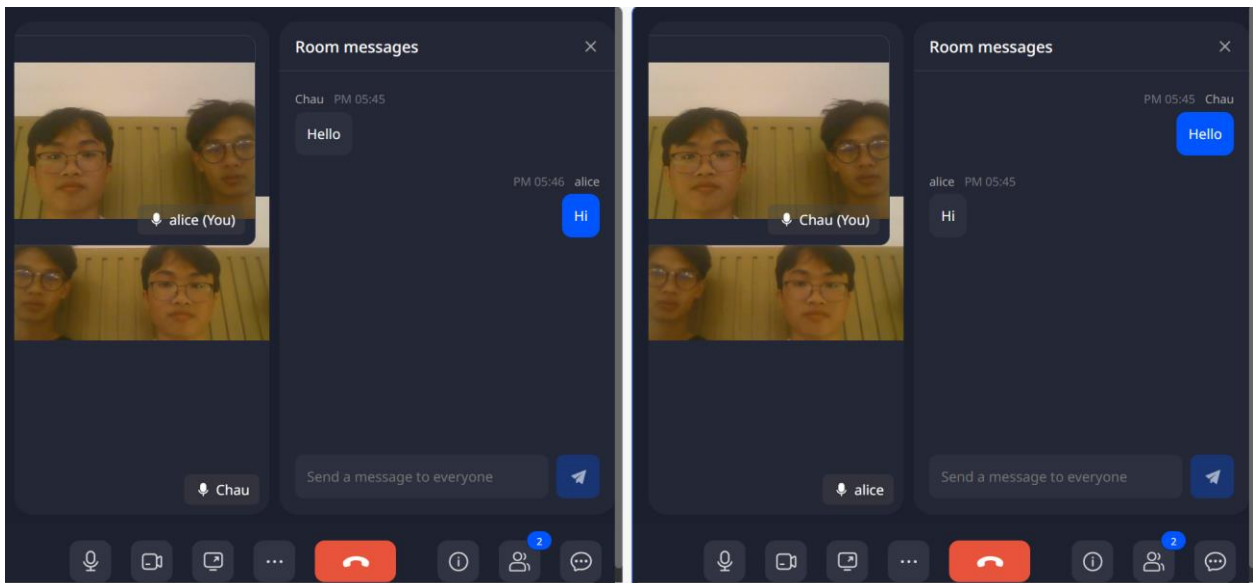
Hình 19. Giao diện trang cá nhân người dùng

Khi click vào “Gọi điện”, giao diện video sẽ hiển thị như sau và chúng ta có thể dễ dàng trao đổi, gặp mặt dù ở bất cứ đâu.



Hình 20. Tính năng Video Call

Ngoài ra, những người tham gia vào cuộc gọi video còn có thể tham gia nhắn tin, phù hợp cho các cuộc họp của một nhóm người với mỗi nhu cầu sử dụng khác nhau.



Hình 21. Tính năng nhắn tin giữa cuộc gọi

TÀI LIỆU THAM KHẢO

- [1] *Spring Boot*. (2024). Spring Boot. <https://spring.io/projects/spring-boot>
- [2] *Postman documentation overview | Postman Learning Center*. (2023, October 19). Postman Learning Center. <https://learning.postman.com/docs/introduction/overview/>
- [3] *MongoDB Documentation*. (2024). MongoDB.com. <https://www.mongodb.com/docs/>
- [4] *Java Documentation - Get Started*. (2018). Oracle Help Center. <https://docs.oracle.com/en/java/>
- [5] *Trail: Learning the Java Language (The Java™ Tutorials)*. (2022). Oracle.com. <https://docs.oracle.com/javase/tutorial/java/index.html>
- [6] *ZEGOCLOUD Developer Documentation*. (2022). Zegocloud.com. <https://www.zegocloud.com/docs>
- [7] *Bài giảng lập trình hướng đối tượng*. (2024). Phạm Thị Kim Ngoan.