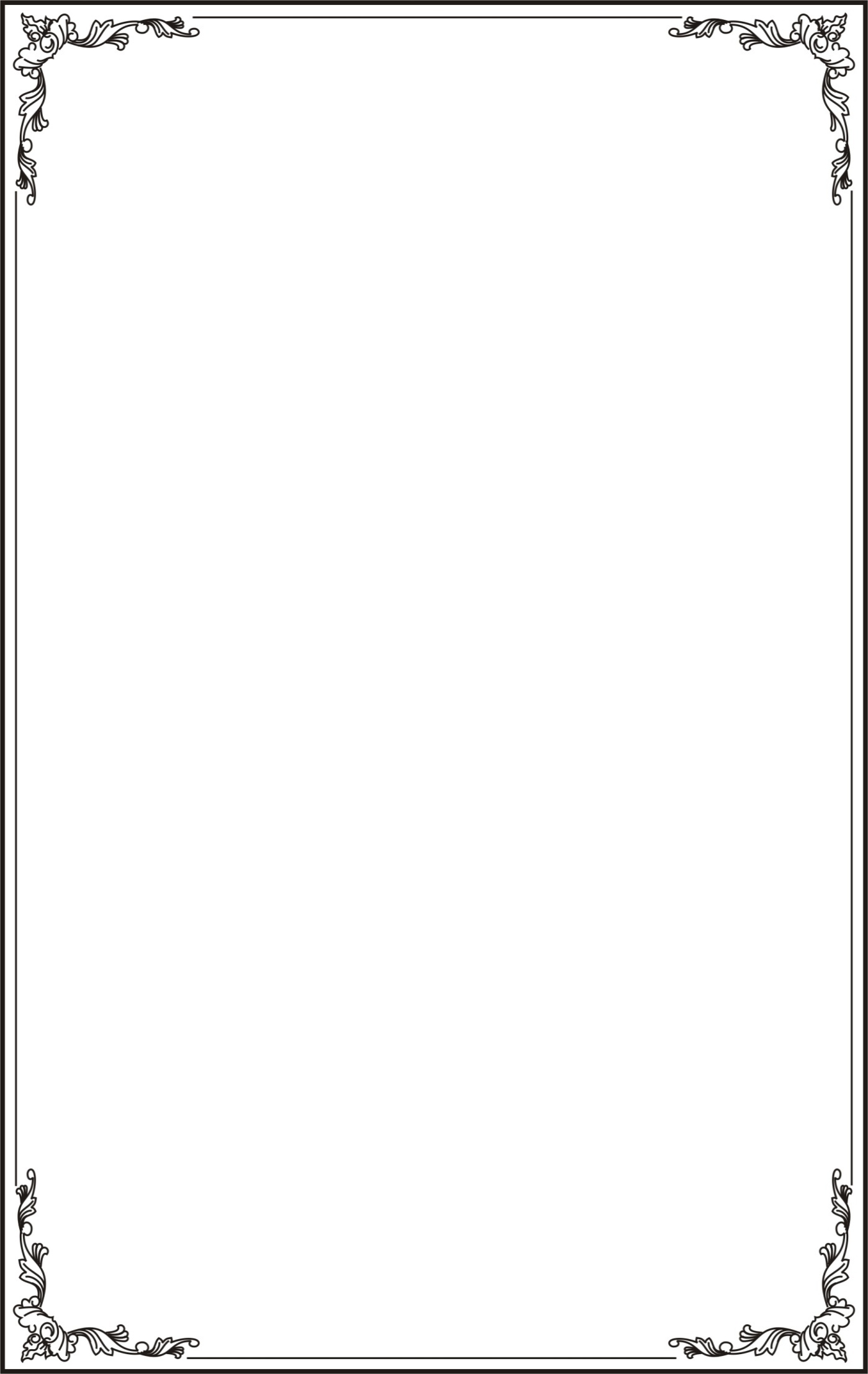
****TRƯỜNG ĐẠI HỌC KIÊN GIANG

**KHOA THÔNG TIN VÀ TRUYỀN THÔNG**

--------------------------------



**LÂM THIỆN TÍNH**

**XÂY DỰNG ỨNG DỤNG NHẮN TIN**

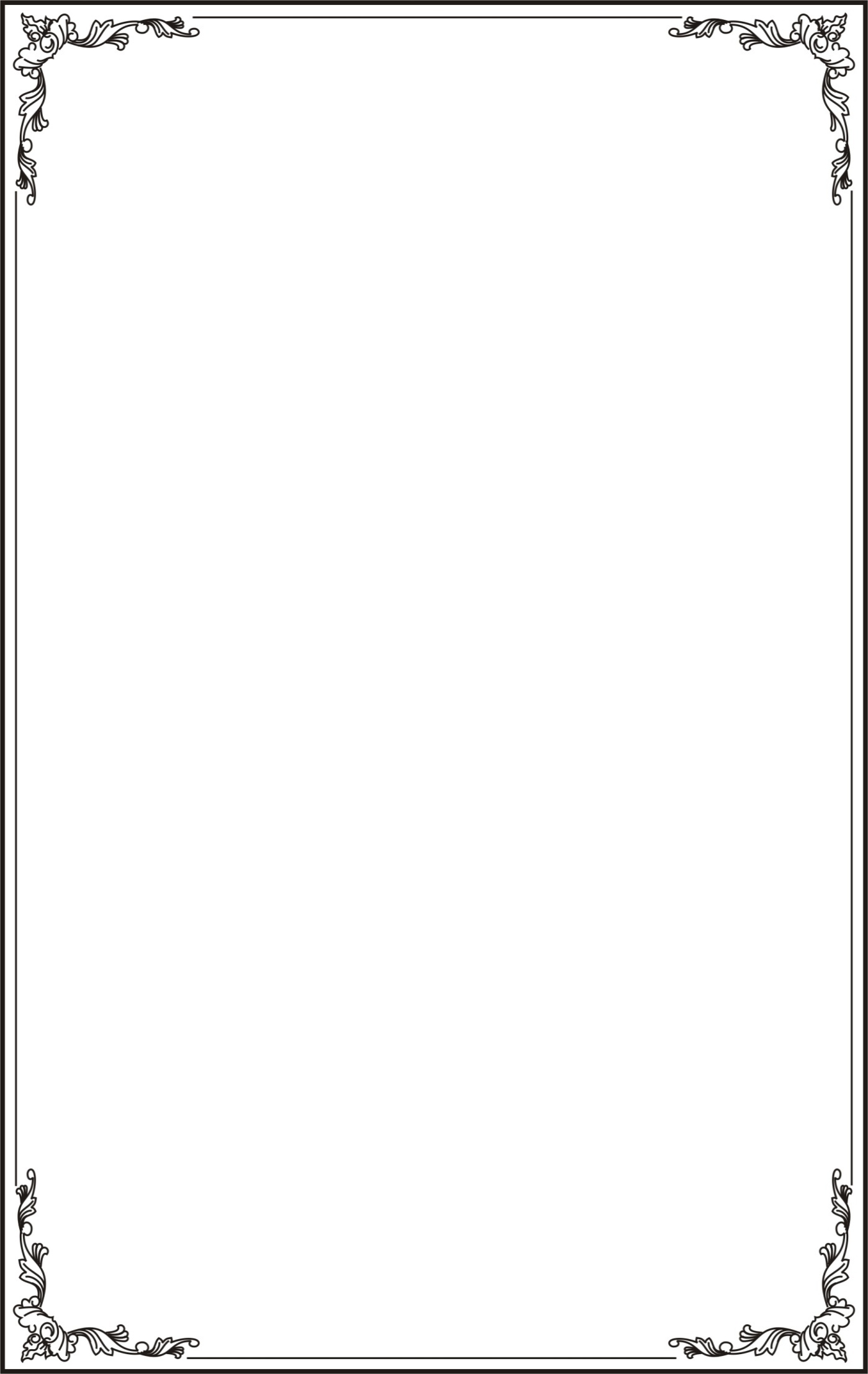
**THỜI GIAN THỰC**

**KHÓA LUẬN TỐT NGHIỆP**

**Ngành: Công nghệ thông tin**

**Mã số ngành: 7480201**

***Kiên Giang - năm 2024***

****TRƯỜNG ĐẠI HỌC KIÊN GIANG

**KHOA THÔNG TIN VÀ TRUYỀN THÔNG**

--------------------------------



**LÂM THIỆN TÍNH**

**MSSV: 2006206072**

**XÂY DỰNG ỨNG DỤNG NHẮN TIN**

**THỜI GIAN THỰC**

**KHÓA LUẬN TỐT NGHIỆP**

**Ngành: Công nghệ thông tin**

**Mã số ngành: 7480201**

**Giảng viên hướng dẫn**

**Ths. NHAN THANH NHÃ**

***Kiên Giang - năm 2024***

**LỜI CẢM ƠN**

Trước hết, tôi xin dành lời cảm ơn chân thành đến quý thầy, cô khoa Thông tin – Truyền thông của trường Đại học Kiên Giang, những người đã dành thời gian và tâm huyết để chia sẻ kiến thức và kinh nghiệm của mình với tôi. Nhờ sự hướng dẫn và sự khuyến khích từ quý thầy, cô, tôi đã có cơ hội nắm vững các kiến thức nền tảng, từ đó có thể tự tin hoàn thiện bài báo cáo này.

Cùng lúc đó, tôi cũng xin bày tỏ lòng biết ơn sâu sắc đến Ths. Nhan Thanh Nhã, người đã dành thời gian và công sức để hướng dẫn và góp ý cho tôi trong suốt quá trình thực hiện khóa luận. Sự chỉ dẫn tận tình của thầy không chỉ giúp tôi hoàn thiện bài báo cáo mà còn giúp tôi phát triển và hoàn thiện bản thân.

Cuối cùng, tôi xin gửi lời cảm ơn đặc biệt đến gia đình và bạn bè, những người luôn ở bên cạnh và động viên, giúp tôi vượt qua khó khăn để hoàn thành bài báo cáo này một cách thành công.

Dù đã cố gắng hoàn thiện tốt nhất trong khả năng của mình, nhưng không thể tránh khỏi những sai sót. Vì vậy, tôi rất mong nhận được sự đóng góp và phản hồi từ quý thầy, cô để có thể bài báo cáo này hoàn thiện hơn.

Một lần nữa xin gửi đến tất cả sự biết ơn chân thành nhất. Chúc trường Đại học Kiên Giang ngày càng phát triển và đào tạo nhiều nhân tài hơn nữa.

*Ngày..... tháng.....năm 2024*

**Sinh viên thực hiện**

*( Ký và ghi rõ họ tên )*

**Lâm Thiện Tính**

**LỜI CAM ĐOAN**

Tôi cam đoan rằng đề tài này là do chính tôi thực hiện, các số liệu thu thập và kết quả phân tích trong đề tài là trung thực, đề tài không trùng với bất kỳ đề tài nghiên cứu khoa học nào.

*Ngày.....tháng.....năm 2024*

**Sinh viên thực hiện**

*( Ký và ghi rõ họ tên )*

**Lâm Thiện Tính**

**NHẬN XÉT CỦA NGƯỜI HƯỚNG DẪN**

*Ngày.... tháng.... năm......*

***Người hướng dẫn***

*( Ký và ghi rõ họ tên )*

**MỤC LỤC**

[CHƯƠNG 1: TỔNG QUAN 16](#_Toc164693021)

[1.1 Lý do chọn đề tài 16](#_Toc164693022)

[1.2 Đối tượng hướng đến 16](#_Toc164693023)

[1.3 Phạm vi đề tài 16](#_Toc164693024)

[1.4 Mục tiêu cần đạt được 16](#_Toc164693025)

[1.4.1 Phi chức năng 16](#_Toc164693026)

[1.4.2 Chức năng 17](#_Toc164693027)

[1.5 Hướng giải quyết và kế hoạch thực hiện 17](#_Toc164693028)

[1.5.1 Hướng giải quyết 17](#_Toc164693029)

[1.5.2 Kế hoạch thực hiện 17](#_Toc164693030)

[CHƯƠNG 2: CƠ SỞ LÝ THUYẾT 19](#_Toc164693031)

[2.1 Phương pháp phân tích và thiết kế 19](#_Toc164693032)

[2.1.1 UML (Unified Modeling Language) 19](#_Toc164693033)

[2.1.2 IoC (Inversion of Control) 22](#_Toc164693034)

[2.1.3 DI (Dependency Injection) 22](#_Toc164693035)

[2.2 Giao thức mạng (Network protocols) 23](#_Toc164693036)

[2.2.1 TCP (Transmission Control Protocol) 23](#_Toc164693037)

[2.2.2 HTTP (Hypertext Transfer Protocol) 25](#_Toc164693038)

[2.2.3 WebSocket 29](#_Toc164693039)

[2.2.4 STOMP (Simple Text Oriented Messaging Protocol) 35](#_Toc164693040)

[2.3 Các công nghệ web 36](#_Toc164693041)

[2.3.1 Công nghệ web phía client 36](#_Toc164693042)

[2.3.1.1 HTML (Hypertext Markup Language) 36](#_Toc164693043)

[2.3.1.2 CSS (Cascading Style Sheets) 38](#_Toc164693044)

[2.3.1.3 Bootstrap 38](#_Toc164693045)

[2.3.1.4 TS (Typescript) 39](#_Toc164693046)

[2.3.1.5 Angular 40](#_Toc164693047)

[2.3.2 Công nghệ web phía server 41](#_Toc164693048)

[2.3.2.1 Java 41](#_Toc164693049)

[2.3.2.2 Spring 42](#_Toc164693050)

[2.4 Cơ sỡ dữ liệu 43](#_Toc164693051)

[2.4.1 SQL (Structured Query Language) 43](#_Toc164693052)

[2.4.2 MySQL 44](#_Toc164693053)

[2.5 ORM (Object-Relational Mapping) 44](#_Toc164693054)

[2.5.1 Khái niệm ORM 44](#_Toc164693055)

[2.5.2 Lý do ra đời 46](#_Toc164693056)

[2.6 JPA (Jakarta/Java Persistence API) 47](#_Toc164693057)

[CHƯƠNG 3: KHẢO SÁT 49](#_Toc164693058)

[3.1 Khảo sát các ứng dụng có sẵn 49](#_Toc164693059)

[3.2 Kết luận 49](#_Toc164693060)

[CHƯƠNG 4: PHÂN TÍCH VÀ THIẾT KẾ 51](#_Toc164693061)

[4.1 Xác định yêu cầu 51](#_Toc164693062)

[4.2 Tìm kiếm các đối tượng hệ thống 51](#_Toc164693063)

[4.3 Xây dựng sơ đồ trường hợp sử dụng (Use case diagram) 52](#_Toc164693064)

[4.3.1 Tác nhân (Actor) 52](#_Toc164693065)

[4.3.2 Trường hợp sử dụng (Use case) 53](#_Toc164693066)

[4.4 Xây dựng sơ đồ tuần tự (Sequence diagram) 71](#_Toc164693067)

[4.4.1 Tiến trình gửi tin nhắn văn bản 71](#_Toc164693068)

[4.4.2 Tiến trình gửi tin nhắn đa phương tiện 72](#_Toc164693069)

[4.5 Xây dựng sơ đồ lớp (Class diagram) 73](#_Toc164693070)

[4.6 Xây dựng các kênh giao tiếp trong STOMP 73](#_Toc164693071)

[4.7 Kiến trúc của ứng dụng 74](#_Toc164693072)

[4.8 Xây dựng sơ đồ triển khai 74](#_Toc164693073)

[CHƯƠNG 5: KẾT QUẢ ĐẠT ĐƯỢC 76](#_Toc164693074)

[5.1 Giao diện đăng nhập 76](#_Toc164693075)

[5.2 Giao diện đăng ký 76](#_Toc164693076)

[5.3 Giao diện trang chủ 77](#_Toc164693077)

[5.4 Giao diện xem tin nhắn 77](#_Toc164693078)

[5.5 Giao diện tạo hội thoại nhóm 78](#_Toc164693079)

[5.6 Giao diện tham gia hội thoại nhóm 79](#_Toc164693080)

[5.7 Giao diện xem trực tiếp file 79](#_Toc164693081)

[5.8 Giao diện các liên hệ 79](#_Toc164693082)

[5.9 Giao diện thêm liên hệ 80](#_Toc164693083)

[5.10 Giao diện yêu cầu liên hệ đã nhận 80](#_Toc164693084)

[5.11 Giao diện yêu cầu liên hệ đã gửi 81](#_Toc164693085)

[5.12 Giao diện đổi mật khẩu 81](#_Toc164693086)

[5.13 Giao diện xóa tài khoản 82](#_Toc164693087)

[5.14 Giao diện xác nhận 82](#_Toc164693088)

[5.15 Giao diện ghi âm 83](#_Toc164693089)

[5.16 Giao diện chọn file 83](#_Toc164693090)

[CHƯƠNG 6: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN 85](#_Toc164693091)

[6.1 Kết luận 85](#_Toc164693092)

[6.2 Hướng phát triển 86](#_Toc164693093)

[TÀI LIỆU THAM KHẢO 87](#_Toc164693094)

**DANH MỤC BIỂU BẢNG**

[Bảng 4.1 Use case sign up 44](#_Toc164330043)

[Bảng 4.2 Use case sign in 44](#_Toc164330044)

[Bảng 4.3 Use case sign out 45](#_Toc164330045)

[Bảng 4.4 Use case view user profile 45](#_Toc164330046)

[Bảng 4.5 Use case update user account 46](#_Toc164330047)

[Bảng 4.6 Use case delete account 47](#_Toc164330048)

[Bảng 4.7 Use case change password 47](#_Toc164330049)

[Bảng 4.8 Use case view contact features 48](#_Toc164330050)

[Bảng 4.9 Use case send contact request 49](#_Toc164330051)

[Bảng 4.10 Use case view recevied contact requests 49](#_Toc164330052)

[Bảng 4.11 Search received contact requests 50](#_Toc164330053)

[Bảng 4.12 Use case accept request 50](#_Toc164330054)

[Bảng 4.13 Use case decline request 51](#_Toc164330055)

[Bảng 4.14 Use case view sent contact request 51](#_Toc164330056)

[Bảng 4.15 Use case search sent contact requests 52](#_Toc164330057)

[Bảng 4.16 Use case delete sent contact requests 52](#_Toc164330058)

[Bảng 4.17 Use case view contacts 53](#_Toc164330059)

[Bảng 4.18 Use case search contacts 53](#_Toc164330060)

[Bảng 4.19 Usecase delete contact 54](#_Toc164330061)

[Bảng 4.20 Use case view conversations 54](#_Toc164330062)

[Bảng 4.21 Use case receive message 55](#_Toc164330063)

[Bảng 4.22 Use case receive notification 55](#_Toc164330064)

[Bảng 4.23 Use case search conversations 55](#_Toc164330065)

[Bảng 4.24 Use case view messages 56](#_Toc164330066)

[Bảng 4.25 Use case search messages 56](#_Toc164330067)

[Bảng 4.26 Use case send text messages 57](#_Toc164330068)

[Bảng 4.27 Use case multimedia message 57](#_Toc164330069)

[Bảng 4.28 Use case record voice 58](#_Toc164330070)

[Bảng 4.29 Use case create group conversation 59](#_Toc164330071)

[Bảng 4.30 Use case update group conversation 60](#_Toc164330072)

[Bảng 4.31 Use case join group conversation 60](#_Toc164330073)

[Bảng 4.32 Use case leave group conversation 61](#_Toc164330074)

**DANH MỤC HÌNH**

[Hình 2.1 Cấu trúc UML 14](#_Toc164331091)

[Hình 2.2 Sơ đồ trường hợp sử dụng 15](#_Toc164331092)

[Hình 2.3 Sơ đồ tuần tự 16](#_Toc164331093)

[Hình 2.4 Sơ đồ lớp 16](#_Toc164331094)

[Hình 2.5 Constructor injection 18](#_Toc164331095)

[Hình 2.6 Setter Injection 18](#_Toc164331096)

[Hình 2.7 Interface injection 19](#_Toc164331097)

[Hình 2.8 Quá trình bắt tay 3 bước 20](#_Toc164331098)

[Hình 2.9 Quá trình bắt tay 4 bước 21](#_Toc164331099)

[Hình 2.10 Cấu trúc của HTTP request 23](#_Toc164331100)

[Hình 2.11 Cấu trúc của HTTP request 24](#_Toc164331101)

[Hình 2.12 Cấu trúc của HTTP request 25](#_Toc164331102)

[Hình 2.13 Cấu trúc của HTTP response 25](#_Toc164331103)

[Hình 2.14 Cấu trúc của WebSocket frame 26](#_Toc164331104)

[Hình 2.15 Luồng hoạt động của WebSocket 28](#_Toc164331105)

[Hình 2.16 Cấu trúc header của HTTP upgrade request/response 29](#_Toc164331106)

[Hình 2.17 Cấu trúc HTTP upgrade request 30](#_Toc164331107)

[Hình 2.18 Cấu trúc HTTP upgrade response 30](#_Toc164331108)

[Hình 2.19 Cấu trúc của STOMP frame 32](#_Toc164331109)

[Hình 2.20 Mã HTML 33](#_Toc164331110)

[Hình 2.21 Mã CSS 34](#_Toc164331111)

[Hình 2.22 Mã Bootstrap 35](#_Toc164331112)

[Hình 2.23 Mã Typescript 36](#_Toc164331113)

[Hình 2.24 Mã Angular 37](#_Toc164331114)

[Hình 2.25 Mã Java 38](#_Toc164331115)

[Hình 2.26 Mã Spring 39](#_Toc164331116)

[Hình 2.27 Mã SQL 40](#_Toc164331117)

[Hình 2.28 Icon MySQL 40](#_Toc164331118)

[Hình 2.29 Metadata của JPA 43](#_Toc164331119)

[Hình 4.1 Một số lớp tìm được 46](#_Toc164331120)

[Hình 4.2 Tác nhân user 47](#_Toc164331121)

[Hình 4.3 Module Authentication 47](#_Toc164331122)

[Hình 4.4 Module User 49](#_Toc164331123)

[Hình 4.5 Module Contact 52](#_Toc164331124)

[Hình 4.6 Module Conversation 59](#_Toc164331125)

[Hình 4.7 Module Group Conversation 63](#_Toc164331126)

[Hình 4.8 Tiến trình gửi tin nhắn văn bản 66](#_Toc164331127)

[Hình 4.9 Tiến trình gửi tin nhắn đa phương tiện 67](#_Toc164331128)

[Hình 4.10 Sơ đồ lớp 68](#_Toc164331129)

[Hình 4.11 Sơ đồ triển khai 69](#_Toc164331130)

[Hình 5.1 Giao diện đăng nhập 70](#_Toc164331131)

[Hình 5.2 Giao diện đăng ký 71](#_Toc164331132)

[Hình 5.3 Giao diện trang chủ 71](#_Toc164331133)

[Hình 5.4 Giao diện xem tin nhắn 72](#_Toc164331134)

[Hình 5.5 Giao diện tạo hội thoại nhóm 72](#_Toc164331135)

[Hình 5.6 Giao diện tham gia hội thoại nhóm 73](#_Toc164331136)

[Hình 5.7 Giao diện đọc file 73](#_Toc164331137)

[Hình 5.8 Giao diện liên hệ 74](#_Toc164331138)

[Hình 5.9 Giao diện thêm liên hệ 74](#_Toc164331139)

[Hình 5.10 Giao diện yêu cầu liên hệ đã nhận 75](#_Toc164331140)

[Hình 5.11 Giao diện yêu cầu liên hệ đã gửi 75](#_Toc164331141)

[Hình 5.12 Giao diện đổi mật khẩu 76](#_Toc164331142)

[Hình 5.13 Giao diện xóa tài khoản 76](#_Toc164331143)

[Hình 5.14 Giao diện xác nhận 77](#_Toc164331144)

[Hình 5.15 Giao diện ghi âm 77](#_Toc164331145)

[Hình 5.16 Giao diện chọn file 78](#_Toc164331146)

**TÓM TẮT**

Trong môi trường sống và làm việc kỹ thuật số hiện nay, giao tiếp hiệu quả là chìa khóa cho cá nhân và tổ chức. Ứng dụng nhắn tin thời gian thực đã trở thành cần thiết, cho phép tương tác mà không bị ràng buộc bởi vị trí địa lý. Báo cáo này trình bày việc phát triển một ứng dụng nhắn tin thời gian thực, sử dụng giao thức WebSocket để đảm bảo giao nhận tin nhắn nhanh chóng. Các tính năng bao gồm đồng bộ hóa tin nhắn thời gian thực, chia sẻ tệp đa phương tiện và thông báo người dùng. Đồng thời, tính năng xác thực người dùng được triển khai để bảo vệ thông tin nhạy cảm. Trong quá trình phát triển, tôi tập trung vào giao diện người dùng (front-end) và máy chủ (back-end) độc lập, sử dụng UML để mô hình hóa hệ thống. Front-end sử dụng Angular với kiến trúc component-based kết hợp với mô hình Observer. Back-end triển khai RESTful APIs và sử dụng mô hình Controller-Service-Repository. Ứng dụng được triển khai trên GitHub Pages (front-end) và Heroku (back-end). Tài liệu hướng dẫn sử dụng và kỹ thuật cũng được tạo ra để hỗ trợ người dùng và các nhà phát triển.

# TỔNG QUAN

## Lý do chọn đề tài

Trong thời đại công nghệ 4.0 hiện nay, giao tiếp hiệu quả là vô cùng quan trọng đối với cá nhân và tổ chức. Điều này đặt ra nhu cầu ngày càng tăng về việc có một phương tiện giao tiếp hiệu quả và nhanh chóng để tương tác và trao đổi thông tin giữa cá nhân và tổ chức.

Ứng dụng nhắn tin thời gian thực không chỉ là một công cụ thông tin, mà còn là một phần không thể thiếu của cuộc sống hàng ngày và môi trường làm việc hiện đại. Đặc biệt, trong doanh nghiệp, sự linh hoạt và tương tác nhanh chóng giữa các bộ phận và phòng ban là yếu tố cực kỳ quan trọng để đảm bảo hiệu suất làm việc và tiến độ dự án.

Việc phát triển một ứng dụng nhắn tin thời gian thực không chỉ giúp đơn giản hóa quá trình giao tiếp mà còn tạo điều kiện cho việc trao đổi thông tin linh hoạt, một cách chính xác và nhanh chóng. Đồng thời, việc sử dụng công nghệ thông tin trong giao tiếp cũng giúp nâng cao tính minh bạch và hiệu quả trong việc quản lý thông tin.

## Đối tượng hướng đến

Đối tượng hướng đến của ứng dụng nhắn tin thời gian thực là các cá nhân, tổ chức, doanh nghiệp có nhu cầu tương tác và trao đổi thông tin nhanh chóng, cũng như quản lý các tin nhắn một cách dễ dàng và hiệu quả.

## Phạm vi đề tài

Đề tài này trình bày việc nghiên cứu, phát triển và triển khai một ứng dụng nhắn tin thời gian thực để tạo điều kiện giao tiếp mượt mà giữa các người dùng.

Ứng dụng cho phép người dùng tham gia vào các cuộc hội thoại nhóm hoặc hội thoại cá nhân, quản lý thông tin liên hệ như yêu cầu liên hệ đã nhận/gửi, nhận và gửi nhiều loại tin nhắn khác nhau như: tin nhắn dạng văn bản, hình ảnh, tập tin...Thông báo đến người dùng khi có tin nhắn mới, hỗ trợ thu âm, gọi điện trực tuyến...

## Mục tiêu cần đạt được

### Phi chức năng

* Giao diện đơn giản dễ sử dụng.
* Dễ dàng bảo trì, mở rộng hệ thống.
* Hoạt động mượt mà trên các môi trường và điều kiện kết nối mạng khác nhau.

### Chức năng

* Đăng ký tài khoản.
* Xác thực và phân quyền người dùng.
* Quản lý thông tin tài khoản, thông tin người dùng.
* Quản lý các cuộc hội thoại.
* Quản lý các liên hệ, yêu cầu liên hệ đến và yêu cầu liên hệ đi.
* Nhận và gửi tin nhắn, hình ảnh, video, tập tin...
* Gửi thông báo tin nhắn mới.
* Ghi âm giọng nói.
* Gọi điện trực tuyến.

## Hướng giải quyết và kế hoạch thực hiện

### Hướng giải quyết

Xây dựng một ứng dụng web nhắn tin thời gian thực. Ứng dụng này thuộc loại SPA. Trong đó, phần client sẽ sử dụng HTML, CSS, TS, Bootstrap và Angular framework để xây dựng giao diện người dùng, phần server sẽ sử dụng ngôn ngữ Java với Spring framework để xử lý các chức năng chính của ứng dụng.

### Kế hoạch thực hiện

1. Tham khảo giao diện và tính năng của một số ứng dụng trò chuyện, nhắn tin hiện nay như: Zalo, Whatssap, Messenger, Telegram...
2. Tham khảo mã nguồn mẫu một số project trên Github.
3. Tìm hiểu kiến trúc component-based, mô hình Observer, giao thức Websocket và STOMP.
4. Tìm hiểu kỹ thuật ORM và JPA.
5. Phân tích chức năng, thiết kế sơ đồ use case và viết scenarios để mô tả các chức năng, sự kiện cũng như tính huống xảy ra trong ứng dụng.
6. Tìm kiếm các đối tượng của ứng dụng tương ứng với các đối tượng thế giới thực. Thiết kế sơ đồ lớp để mô tả cấu trúc của ứng dụng.
7. Thiết kế sơ đồ tuần tự mô tả sự tương tác giữa các đối tượng trong ứng dụng.
8. Viết mã.
9. Triển khai ứng dụng lên máy chủ thực tế.
10. Xây dựng tài liệu kỹ thuật và tài liệu hướng dẫn sử dụng.

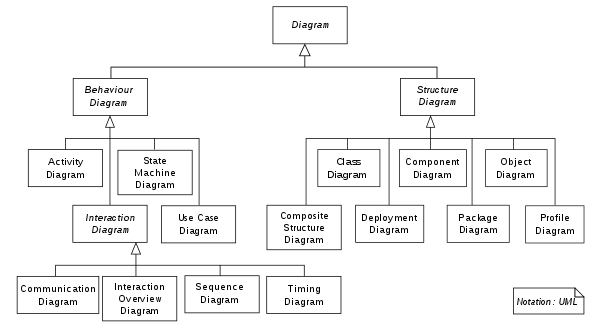
# CƠ SỞ LÝ THUYẾT

## Phương pháp phân tích và thiết kế

### UML (Unified Modeling Language)

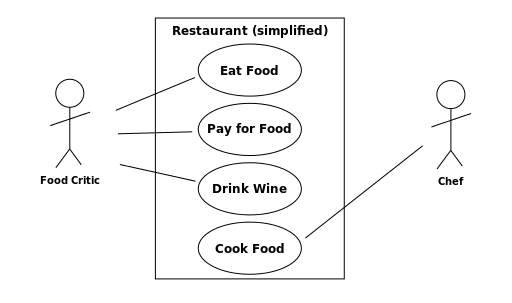
UML (Unified Modeling Language - Ngôn ngữ Mô hình hóa Thống nhất). Đây là một ngôn ngữ mô hình hóa phổ biến được sử dụng trong việc phát triển phần mềm, dùng để mô tả, thiết kế các hệ thống phần mềm.

UML cung cấp ký hiệu chuẩn cho nhiều loại sơ đồ, có thể tạm chia thành ba nhóm chính: Behaviour Diagram (sơ đồ hành vi), Interaction Diagram (sơ đồ tương tác) và Structure Diagram (sơ đồ cấu trúc).



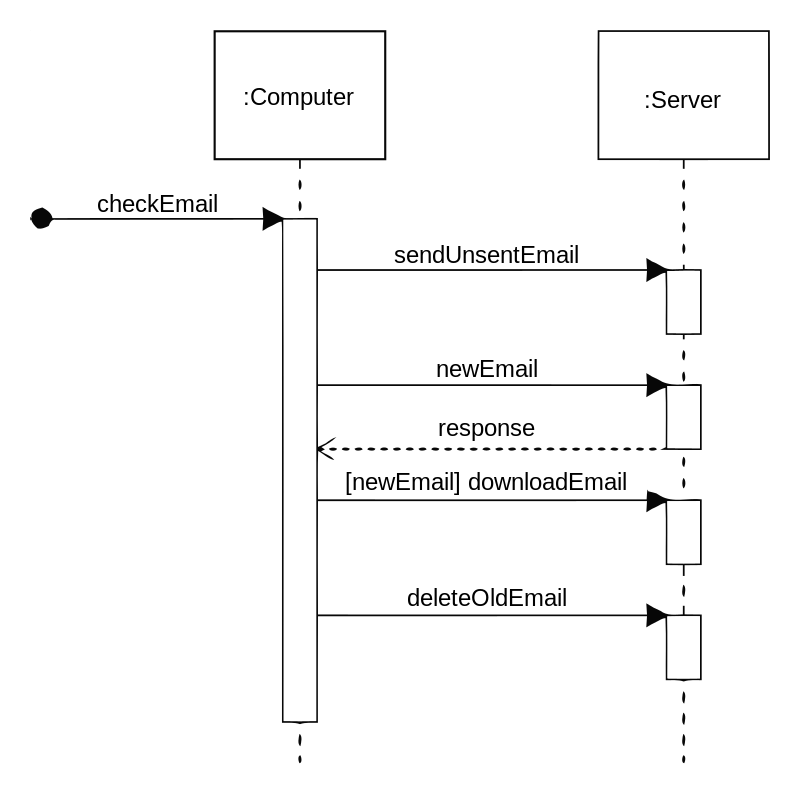
Hình 2.1 Cấu trúc UML

Sơ đồ hành vi thể hiện khía cạnh động của hệ thống. Nó nhấn mạnh những gì phải xảy ra trong hệ thống đang được mô hình hóa. Vì sơ đồ hành vi minh họa hành vi của một hệ thống nên chúng được sử dụng rộng rãi để mô tả chức năng của hệ thống phần mềm. Ví dụ: Use case (sơ đồ ca sử dụng) mô tả các chức năng của hệ thống từ quan điểm của người sử dụng (người dùng cuối) và các tương tác giữa hệ thống và người dùng.



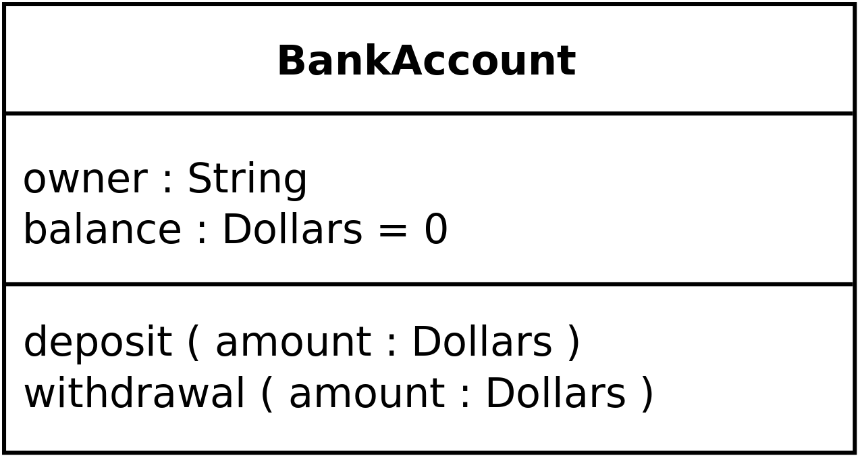
Hình 2.2 Sơ đồ trường hợp sử dụng

Sơ đồ tương tác, một tập hợp con của sơ đồ hành vi, nhấn mạnh luồng điều khiển và dữ liệu giữa những thứ trong hệ thống được mô hình hóa. Ví dụ, Sequence Diagram (sơ đồ tuần tự) cho thấy cách các đối tượng giao tiếp với nhau về một chuỗi các thông điệp.



Hình 2.3 Sơ đồ tuần tự

Sơ đồ cấu trúc thể hiện các khía cạnh tĩnh của hệ thống. Nó nhấn mạnh những thứ phải có trong hệ thống được mô hình hóa. Vì sơ đồ cấu trúc thể hiện cấu trúc nên chúng được sử dụng rộng rãi trong việc ghi lại kiến trúc phần mềm của hệ thống phần mềm. Ví dụ: Class Diagram (sơ đồ lớp) mô tả cấu trúc tĩnh của hệ thống phần mềm bằng cách biểu diễn các lớp, các thuộc tính của chúng, các phương thức và mối quan hệ giữa các lớp.



Hình 2.4 Sơ đồ lớp

Mặc dù UML vẫn là một công cụ quan trọng trong việc phát triển phần mềm, nhưng có một số lý do mà sự sử dụng của nó có thể giảm đi hoặc không còn được ưa chuộng như trước:

* Sự nổi lên của các phương pháp linh hoạt như Agile đã chuyển trọng tâm sang các phương pháp tiếp cận nhẹ nhàng và lặp đi lặp lại. Agile nhấn mạnh đến phần mềm hoạt động được hơn là tài liệu toàn diện, điều này đôi khi dẫn đến việc giảm bớt sự chú trọng vào mô hình hóa UML chi tiết.
* Trong các dự án nhỏ hoặc khi các thành viên trong nhóm làm việc cùng nhau trực tiếp, việc sử dụng UML có thể trở nên không cần thiết. Thay vào đó, các cuộc thảo luận trực tiếp và giao tiếp trực tiếp có thể được ưa chuộng hơn.
* Một số công cụ và ngôn ngữ mới đã xuất hiện, cung cấp các cách tiếp cận khác cho việc mô hình hóa và thiết kế hệ thống. Các công nghệ như Domain-Driven Design (DDD), Behavior-Driven Development (BDD)…

### IoC (Inversion of Control)

IoC (Inversion of Control – Đảo ngược điều khiển) là một nguyên lý thiết kế trong lập trình hướng đối tượng. IoC giúp giảm sự phụ thuộc của các đối tượng với nhau bằng cách di chuyển việc khởi tạo đối tượng từ lớp sử dụng đối tượng sang một container hoặc framework.

Trong IoC, các đối tượng định nghĩa phần phụ thuộc của chúng, trong khi framework quản lý vòng đời của các đối tượng này. Điều này ngược lại với cách lập trình truyền thống, nơi mà đối tượng tự quản lý các phụ thuộc của chính nó.

Một số đặc điểm chính của IoC:

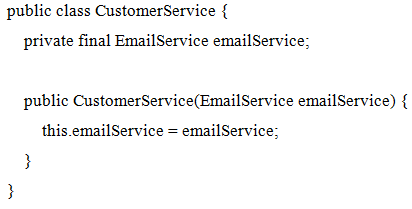
* Các đối tượng không tự khởi tạo các phụ thuộc của chúng, mà được cung cấp các phụ thuộc này bởi một container hoặc framework.
* Các đối tượng chỉ đơn giản khai báo các phụ thuộc của chúng, không quan tâm đến cách thức tạo ra chúng.
* Việc quản lý vòng đời của các đối tượng và các phụ thuộc được thực hiện bởi framework, không phải bởi chính các đối tượng.

### DI (Dependency Injection)

DI (Dependency Injection – Tiêm phụ thuộc) là một kỹ thuật lập trình trong đó đối tượng nhận được các đối tượng phụ thuộc của nó từ bên ngoài thay vì tự tạo ra chúng. Đây là một cách cụ thể để thực hiện nguyên tắc IoC.

Có 3 cách chính để thực hiện Dependency Injection:

1. Constructor Injection: Các phụ thuộc được truyền vào qua constructor của lớp.



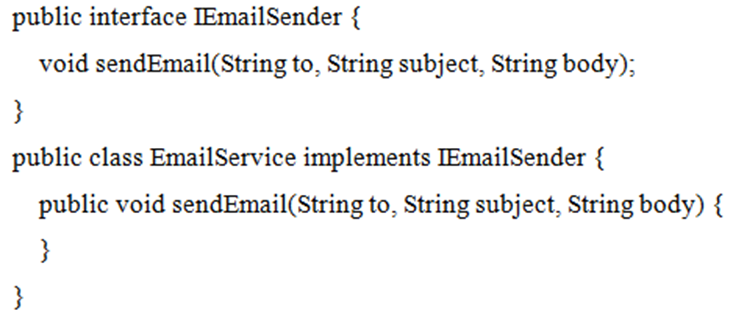
Hình 2.5 Constructor injection

1. Setter Injection: Các phụ thuộc được thiết lập thông qua các phương thức setter.



Hình 2.6 Setter Injection

1. Interface Injection: Các phụ thuộc được truyền vào thông qua một interface



Hình 2.7 Interface injection

## Giao thức mạng (Network protocols)

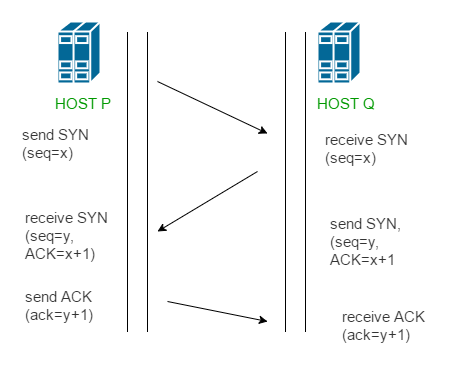
### TCP (Transmission Control Protocol)

TCP là một trong những giao thức chính của bộ giao thức Internet. Nó xuất phát từ việc triển khai ban đầu của mạng, trong đó nó bổ sung cho Giao thức Internet (IP). Do đó, toàn bộ bộ giao thức này thường được gọi là TCP/IP. TCP cung cấp việc truyền tải đáng tin cậy, được sắp xếp và kiểm tra lỗi của một luồng các octet (byte) giữa các ứng dụng đang chạy trên các máy chủ giao tiếp qua một mạng IP.

TCP chia dữ liệu thành các gói tin (segments) nhỏ hơn trước khi truyền. Máy nhận sẽ tái lắp ráp các gói tin thành dữ liệu gốc.

TCP đảm bảo dữ liệu được truyền đến đích một cách đầy đủ và đúng thứ tự. Nó sử dụng các cơ chế như checksum, xác nhận (acknowledgement) và retransmission để xử lý các gói tin bị mất hoặc hỏng.

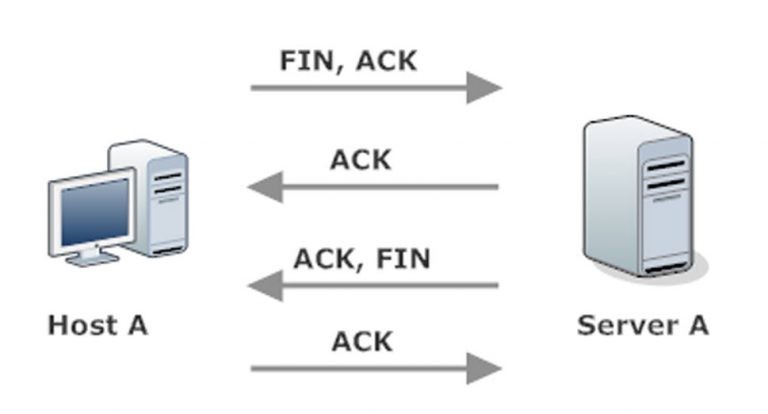
TCP là một giao thức hướng kết nối, TCP thiết lập và duy trì một kết nối logic giữa hai máy trước khi truyền dữ liệu. Quá trình thiết lập kết nối được thực hiện thông qua bắt tay 3 bước (three-way handshake):



Hình 2.8 Quá trình bắt tay 3 bước

1. Client gửi một gói tin SYN (Synchronize) tới server để yêu cầu thiết lập kết nối. Gói tin này chứa một số sequence number được sinh ngẫu nhiên.
2. Khi server nhận được gói tin SYN, nó sẽ trả lời bằng một gói tin SYN-ACK(Synchronize-Acknowledge). Gói tin này chứa sequence number của server và xác nhận (ACK) sequence number của client.
3. Cuối cùng, client sẽ gửi lại một gói tin ACK (Acknowledge) để xác nhận rằng nó đã nhận được gói tin SYN-ACK từ server. Ở bước này, kết nối TCP đã được thiết lập và sẵn sàng để truyền dữ liệu.

Sau khi truyền dữ liệu, quá trình đóng kết nối được thực hiện thông qua bắt tay 4 bước (four-way handshake):



Hình 2.9 Quá trình bắt tay 4 bước

1. Một trong hai bên (ví dụ là client) gửi FIN (Finish) flag đến server để thông báo rằng muốn kết thúc kết nối.
2. Bên server nhận được FIN flag và trả lời bằng một ACK flag để xác nhận rằng nó đã nhận được thông báo kết thúc kết nối. Tại thời điểm này, kết nối chỉ bị chấm dứt từ phía client vì FIN chỉ được gửi từ client và nó đã nhận được ACK từ server nhưng server vẫn mở kết nối TCP vì server chưa gửi FIN. Trạng thái này được gọi là trạng thái FIN\_WAIT\_2 vì client mong đợi cũng sẽ nhận được FIN từ server.
3. Server gửi một FIN-ACK flag đến client
4. Client gửi một ACK flag đến server để xác nhận đã nhận được FIN flag từ server. Khi client gửi ACK flag, nó chuyển sang trạng thái TIME\_WAIT, thời gian của trạng thái TIME\_WAIT tùy vào cách triển khai, nhưng giá trị điển hình của chúng là 30 giây, 1 phút và 2 phút. Sau khi chờ đợi, kết nối chính thức đóng lại và tất cả tài nguyên ở phía máy khách (bao gồm số cổng và dữ liệu bộ đệm) sẽ được giải phóng.

### HTTP (Hypertext Transfer Protocol)

HTTP là giao thức tầng ứng dụng trong mô hình OSI (Open Systems Interconnection) hoặc TCP/IP (Transmission Control Protocol/Internet Protocol) để truyền các tài liệu siêu phương tiện, chẳng hạn như HTML. Nó được thiết kế để liên lạc giữa trình duyệt web và máy chủ web, nhưng nó cũng có thể được sử dụng cho các mục đích khác.

HTTP tuân theo mô hình máy yêu cầu / đáp ứng(request/response), trong đó máy khách (client) mở kết nối để gửi request, sau đó đợi cho đến khi nhận được response từ máy chủ (server). HTTP là giao thức không trạng thái (stateless), nghĩa là server không giữ bất kỳ dữ liệu (trạng thái) nào giữa hai yêu cầu.

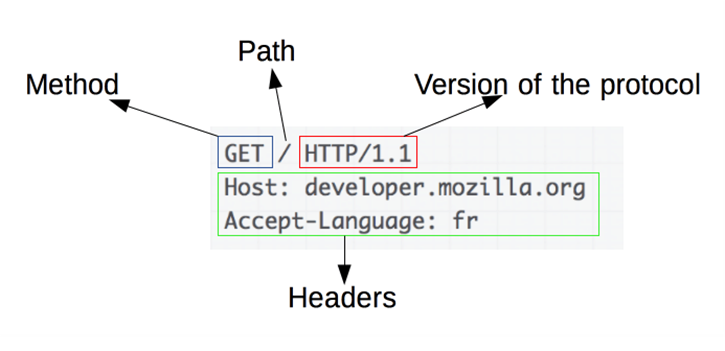
Một kết nối được kiểm soát ở tầng giao vận (transport layer trong mô hình mạng) và do đó về cơ bản nằm ngoài phạm vi của HTTP. HTTP không yêu cầu giao thức truyền tải cơ bản phải dựa trên kết nối; nó chỉ yêu cầu nó phải đáng tin cậy hoặc không bị mất tin nhắn (ít nhất là có lỗi trong những trường hợp như vậy). Trong số hai giao thức truyền tải phổ biến nhất trên Internet, TCP là đáng tin cậy còn UDP thì không. Do đó, HTTP dựa trên tiêu chuẩn TCP.

Trước khi client và server có thể trao đổi cặp HTTP request/response, chúng phải thiết lập kết nối TCP, một quá trình yêu cầu nhiều lượt khứ hồi. Hành vi mặc định của HTTP/1.0 là mở một kết nối TCP riêng cho từng cặp request/response HTTP. Điều này kém hiệu quả hơn việc chia sẻ một kết nối TCP khi nhiều yêu cầu được gửi liên tiếp. Để giảm thiểu hạn chế này, HTTP/1.1 đã giới thiệu pipelining (được chứng minh là khó triển khai) và các kết nối liên tục: kết nối TCP cơ bản có thể được kiểm soát một phần bằng cách sử dụng Connection header: Connection: keep-alive yêu cầu server giữ kết nối TCP sau khi response để client có thể gửi thêm request mới, Connection: close để yêu cầu server đóng kết nối sau khi response.

Client và server giao tiếp bằng cách trao đổi các message riêng lẻ. Các messages được gửi bởi client, thường là trình duyệt Web, được gọi là request và các messages được gửi bởi server dưới dạng câu trả lời được gọi là response.

Một request gồm các thành phần sau: request line (dòng yêu cầu), headers (các tiêu đề) và body (thân hay nội dung):

* Request line bao gồm các HTTP methods (phương thức HTTP) như GET, POST, PATCH, DELETE…URL (Uniform Resource Locator) của tài nguyên và phiên bản HTTP được sử dụng
* Headers cung cấp các thông tin bổ sung về request ví dụ như Content-Type, Accept-Language, cookie…
* Body chứa dữ liệu bổ sung được gửi đến server, chẳng hạn như dữ liệu mẫu hoặc dữ liệu được tải lên.



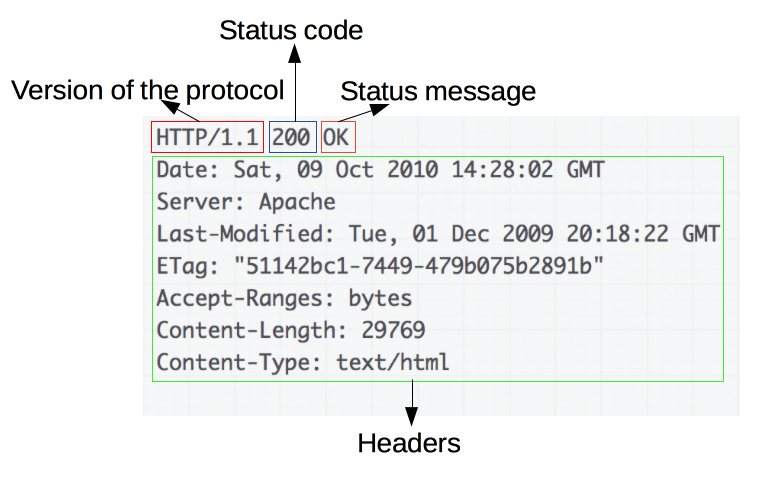
Hình 2.10 Cấu trúc của HTTP request

Trong đó:

* Method là một HTTP method thường là một động từ như GET, POST hoặc một danh từ như OPTIONS hoặc HEAD để xác định thao tác mà client muốn thực hiện. Thông thường, client muốn tìm nạp tài nguyên (sử dụng GET) hoặc đăng giá trị của biểu mẫu HTML (sử dụng POST), mặc dù có thể cần nhiều thao tác hơn trong các trường hợp khác.
* Path là đường dẫn của tài nguyên cần tìm nạp.
* Version of the protocol là phiên bản của giao thức HTTP.
* Headers là các tiêu đề tùy chọn truyền tải thông tin bổ sung cho máy chủ.
* Body là nội dung, đối với một số phương thức như POST dùng để chứa tài nguyên được gửi.

Một response bao gồm các thành phần sau: status line (dòng trạng thái), headers và body:

* Status line bao gồm phiên bản HTTP được sử dụng, status code (mã trạng thái) như 200 OK, 404 NOT FOUND…và thông báo trạng thái.
* Headers cung cấp thông tin bổ sung về response ví dụ như Content-Type, Last-Modified…
* Body chứa dữ liệu trả về từ server, chẳng hạn như nội dung trang web, hình ảnh hoặc video…



Hình 2.11 Cấu trúc của HTTP request

Trong đó:

* Version of the HTTP protocol tương tự như request.
* Status code là mã trạng thái, cho biết yêu cầu có thành công hay không và tại sao. Mã trạng thái được nhóm thành 5 lớp: Information responses (100 -> 199), Successful responses (200 -> 299), Redirection messages (300 -> 399), Client error responses (400 -> 499), Server error responses (500 -> 599). Mỗi lớp bao gồm nhiều mã trạng thái dùng để biểu thị các trạng thái khác nhau của request.
* Status message là thông báo trạng thái, mô tả ngắn gọn không chính thống về mã trạng thái.
* Headers tương tự như request.
* Body là phần nội dung tùy chọn chứa tài nguyên được tìm nạp.

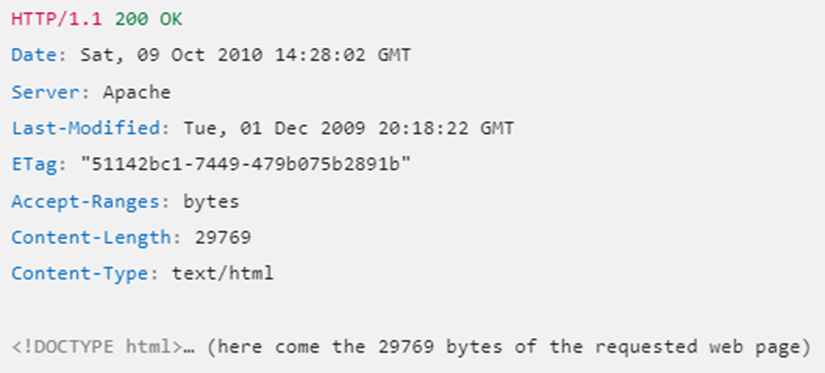
Luồng hoạt động của HTTP:

1. Mở kết nối TCP: Kết nối TCP được sử dụng để gửi một hoặc nhiều requests và nhận responses. Client có thể mở một kết nối mới, sử dụng lại kết nối hiện có hoặc mở một số kết nối TCP tới máy chủ.
2. Gửi tin nhắn HTTP: Tin nhắn HTTP (trước HTTP/2) có thể đọc được. Với HTTP/2, những thông báo đơn giản này được gói gọn trong các khung khiến chúng không thể đọc trực tiếp được nhưng nguyên tắc vẫn giữ nguyên. Ví dụ:



Hình 2.12 Cấu trúc của HTTP request

1. Đọc phản hồi được gửi bởi máy chủ, chẳng hạn như:



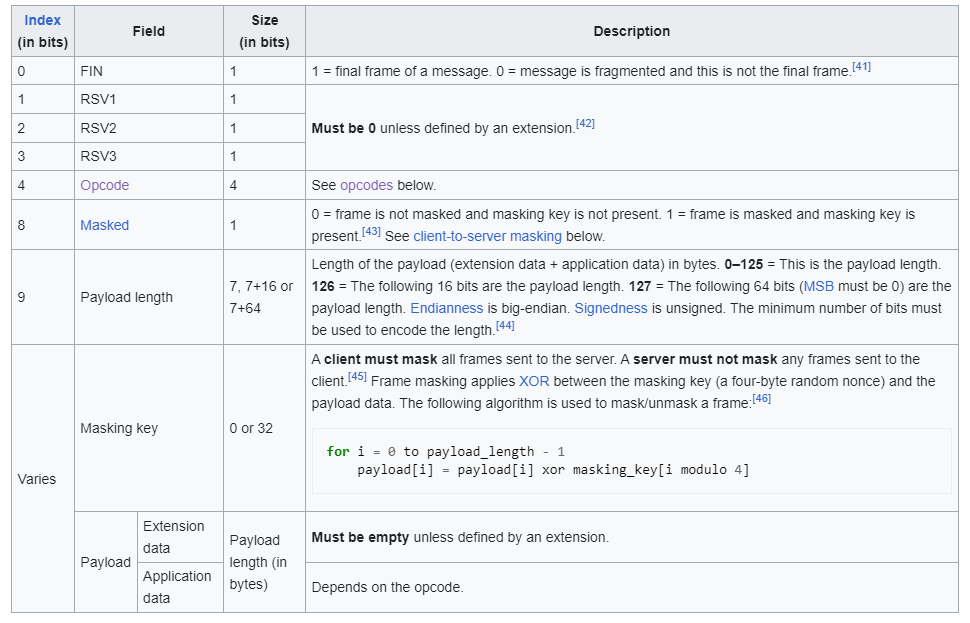
Hình 2.13 Cấu trúc của HTTP response

1. Đóng hoặc sử dụng lại kết nối cho các yêu cầu tiếp theo.

### WebSocket

WebSocket là một giao thức tầng ứng dụng, cung cấp các kênh liên lạc hai chiều đồng thời qua một kết TCP duy nhất. Nó giúp thiết lập một kết nối tương tác liên tục giữa trình duyệt (client) và máy chủ (server) để truyền và nhận dữ liệu mà không cần phải thực hiện các yêu cầu HTTP mới mỗi lần cần truyền thông tin, ngoài ra còn cho phép các giao thức khác hoạt động trên nó.

Sau khi kết nối được thiết lập, giao tiếp sẽ chuyển sang giao thức binary frame-based không tuân thủ giao thức HTTP. Cấu trúc của một WebSocket frame:

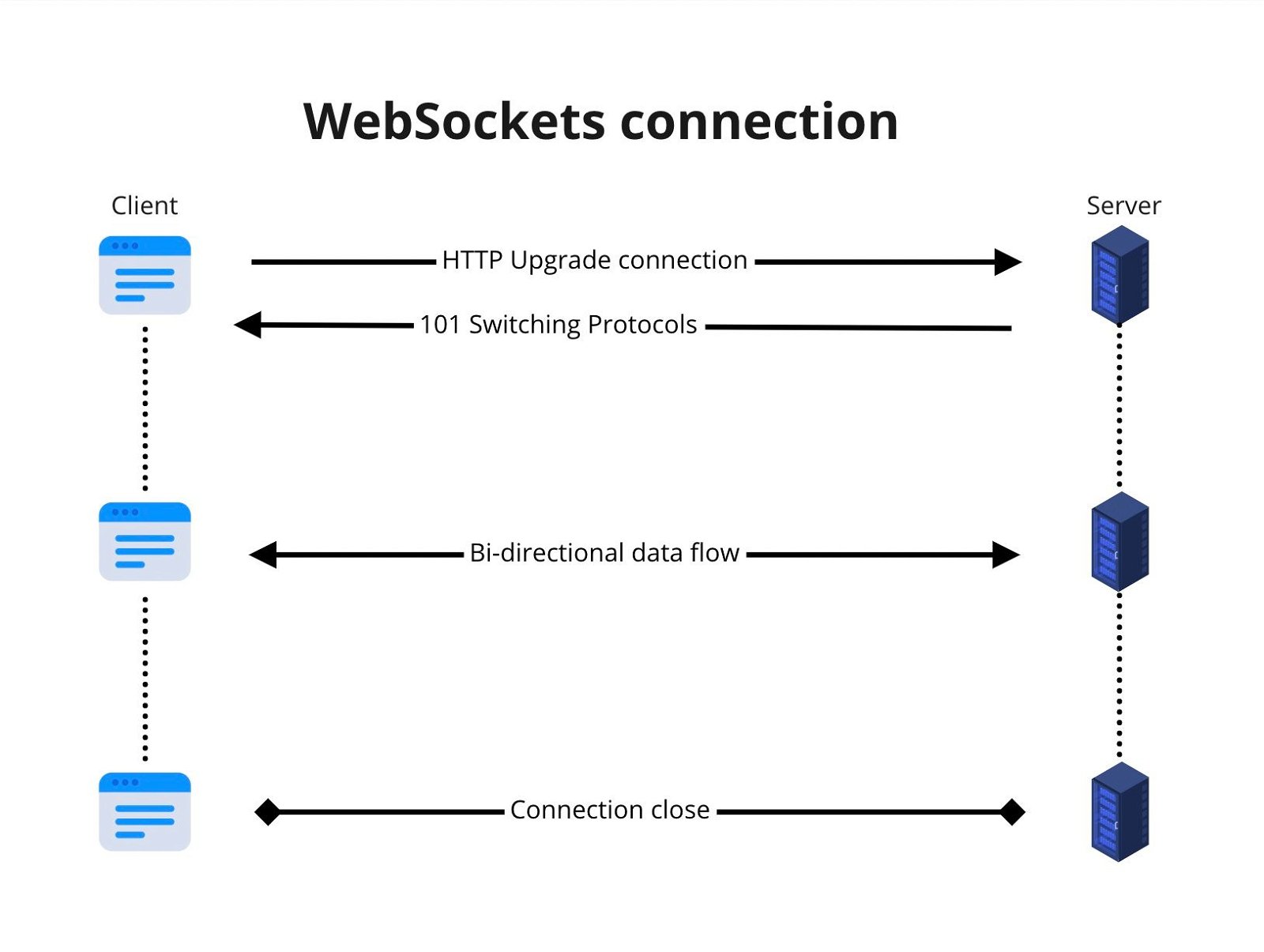


Hình 2.14 Cấu trúc của WebSocket frame

Trong đó:

* “FIN“ (index = 0, size = 1-bit) giúp xác định xem frame hiện tại có phải là frame cuối cùng của một message (gồm nhiều frames trong trường hợp message có kích thước lớn) hay không. Với FIN = 1 đại diện đây là frame cuối cùng của một message, FIN = 0 đại diện còn có các frame khác sau nó trong cùng một message.
* “RSV1“, “RSV2“, “RSV3“ (index = [1, 2, 3], size = [1, 1, 1]-bit) dành cho mục đích mở rộng (RSV - Reserved Bits). Mỗi bit đại diện cho một mục đích mở rộng cụ thể. Tuy nhiên, hiện tại các bit này đa số đều không được sử dụng và phải đặt giá trị là 0.
* “OpCode” (index = 4, size = 4-bits) giúp xác định loại frame (Continuation, Data frame hoặc Control frame) cũng như xác định loại dữ liệu thực của Payload, tức là dữ liệu nào đang được gửi hoặc được yêu cầu trong một frame.
  + Continuation
    - OpCode 0x0: frame hiện tại đang chứa dữ liệu tiếp theo của một message đã được phân mảnh.
  + Data frame
    - OpCode 0x1: Đây là Text frame, chứa dữ liệu văn bản. Payload của frame này sẽ là chuỗi ký tự Unicode (UTF-8).
    - OpCode 0x2: Đây là Binary frame, chứa dữ liệu nhị phân. Payload của frame này có thể là hình ảnh, âm thanh, video hoặc bất kỳ dữ liệu nhị phân nào khác.
  + Control frame
    - OpCode 0x8: Đây là Close frame, được sử dụng để đóng kết nối WebSocket hoặc thông báo lỗi.
    - OpCode 0x9: Đây là Ping frame, được sử dụng để kiểm tra kết nối và gửi ping tới đối tác.
    - OpCode 0xA: Đây là Pong frame, được sử dụng để phản hồi lại ping từ đối tác.
* “Masked“ (index = 8, size = 1-bit) giúp xác định liệu Payload của frame có được mã hóa hay không. Với Masked = 1 đại diện cho Payload của frame đã được mã hóa, với Masked = 0 đại diện cho Payload của frame không được mã hóa.
* “Payload length” (index = 9, size = 7 | 7+16 | 7+64-bits) giúp xác định kích thước của Payload.
  + Nếu giá trị của Payload length là từ 0 đến 125 (7-bit) thì trường Payload length sẽ chứa giá trị trực tiếp, không cần thêm bất kỳ byte nào khác.
  + Nếu giá trị của Payload length là từ 126 đến 65535 (16-bit): Trong trường Payload length, 7-bit đầu tiên sẽ có giá trị 126 (1111110), sau đó 16-bit tiếp theo sẽ được sử dụng để chỉ ra kích thước thực của Payload.
  + Nếu giá trị của Payload length là từ 65536 trở lên (64-bit): Tương tự, 7-bit đầu tiên trong trường Payload length sẽ có giá trị 127 (1111111), sau đó 64-bit tiếp theo sẽ được sử dụng để chỉ ra kích thước thực của Payload.
* “Masking key” (size = 0 | 32-bits) giá trị này phụ thuộc vào trường Masked.
  + Nếu Masked = 0, không có Masking key được sử dụng và Payload được gửi đi một cách trực tiếp, không thông qua một phép mã hóa nào.
  + Nếu Masked = 1, thực hiện web phép XOR (hoặc AND) giữa Masking key và payload để tạo ra Payload đã được masked trước khi gửi đi. Người nhận frame sẽ sử dụng lại Masking key để giải mã Payload và đọc dữ liệu đúng đắn.
* “Payload” (size dựa vào kích thước thực của payload) phần dữ liệu thực sự mà frame chứa và truyền đi.

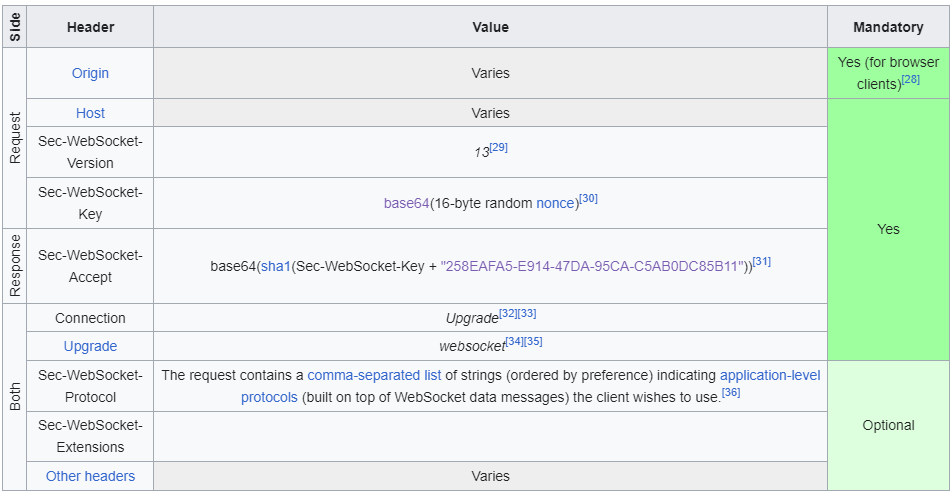
Luồng hoạt động:



Hình 2.15 Luồng hoạt động của WebSocket

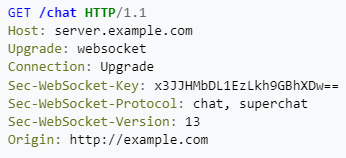
1. Clien gửi một HTTP request đặc biệt để yêu cầu thiết lặp kết nối. Request này dùng GET method với HTTP version từ 1.1 trở lên, request này cũng bao gồm các headers như: Connection, Sec-WebSocket-Key, Sec-WebSocket-Version...
2. Server nhận request, server sẽ kiểm tra và xác thực các headers. Nếu hợp lệ, server sẽ trả về một HTTP response với status code 101 (Switching Protocols), response này cũng bao gồm các headers như: Upgrade, Connection, Sec-WebSocket-Accept… để hoàn thành quá trình bắt tay.
3. Sau khi bắt tay thành công, kết nối TCP được "nâng cấp" thành kết nối WebSocket. Client và server có thể bắt đầu trao đổi dữ liệu thông qua các frame WebSocket.
4. Để ngăn kết nối bị đóng do timeout, WebSocket sử dụng các Control frame như Ping frame và Pong frame để duy trì kết nối.
5. Khi muốn đóng kết nối, client hoặc server gửi một Close frame để thông báo và đóng kết nối.

Cấu trúc của header của request/response:



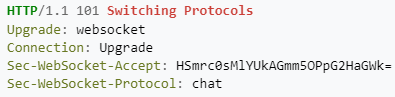
Hình 2.16 Cấu trúc header của HTTP upgrade request/response

Ví dụ một HTTP request để yêu cầu nâng cấp giao thức:



Hình 2.17 Cấu trúc HTTP upgrade request

Ví dụ một HTTP response



Hình 2.18 Cấu trúc HTTP upgrade response

Trong đó:

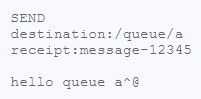
* “Origin” chỉ ra nguồn của request (địa chỉ URL gốc của trang web) giúp ngăn chặn tấn công Cross-Site WebSocket Hijacking (CSWSH), cuộc tấn công này là hình thức kẻ tấn công cố gắng sử dụng kết nối WebSocket từ một trang web khác để tấn công một ứng dụng WebSocket khác.
* “Host’ chỉ ra tên miền của server mà request gửi đến.
* “Sec-WebSocket-Version” chỉ ra phiên bản của WebSocket mà client hỗ trợ, giúp server WebSocket có thể tương thích và hỗ trợ đúng cách.
* “Sec-WebSocket-Key” là một giá trị ngẫu nhiên được tạo ra bởi client Server sẽ sử dụng chuỗi này để tạo ra một mã băm (hash) từ chuỗi đó kết hợp với một chuỗi GUID đã biết trước (ví dụ: "258EAFA5-E914-47DA-95CA-C5AB0DC85B11"). Sau đó, server sẽ trả về một HTTP response với mã hash này trong trường “Sec-WebSocket-Accept”.
* “Connection” quy định các quy tắc kết nối giữa client và server. “Connection: Upgrade” khi client muốn nâng cấp kết nối từ HTTP sang WebSocket.
* “Upgrade” chuyển đổi (upgrade) loại kết nối từ một phiên bản giao thức sang một phiên bản khác. “Upgrade: websocket” chỉ ra rằng client muốn nâng cấp kết nối thành kết nối WebSocket.
* "Sec-WebSocket-Protocol" chỉ định các giao thức con hoặc ứng dụng cụ thể mà client và server đang sử dụng trên kết nối WebSocket.

### STOMP (Simple Text Oriented Messaging Protocol)

STOMP (Simple Text Oriented Messaging Protocol) là một giao thức dựa trên văn bản được thiết kế để truyền tin nhắn không đồng bộ giữa các máy khách thông qua các máy chủ trung gian.

STOMP Frame là đơn vị dữ liệu cơ bản được sử dụng trong giao thức STOMP. Nó là một chuỗi văn bản được chia thành các dòng, mỗi dòng có một cú pháp cụ thể. Một STOMP Frame bao gồm các phần sau: một command, một tập hợp các headers và body tùy chọn:

* Command (dòng lệnh): Là phần bắt buộc đầu tiên của mỗi frame dùng dể xác định hành động mà frame đó muốn thực hiện. Các commands bao gồm: CONNECT, SEND, SUBSCRIBE, UNSUBSCRIBE, ACK, NACK, BEGIN, COMMIT, ABORT, DISCONNECT, MESSAGE, RECEIPT, ERROR. Các commands đều phải được in hoa. Kết thúc của mỗi command là một EOL (end of line - ký tự kết thúc dòng), ký tự này bao gồm một optional CR (carriage return: octet 13) dùng để di chuyển con trỏ đến đầu dòng hiện tại và một required LF (line feed: octet 10) dùng để xuống dòng.
* Headers (Các tiêu đề): bao gồm một cặp giá trị dạng <key>:<value> được ngắn cách bởi dấu 2 chấm, dùng để chứa các thông tin bổ sung cho command. Ví dụ: destination:/queue/myqueue, content-type:text/plain…Mỗi header được kết thúc bởi một EOL. Một khoảng trống (thêm một EOL) sau header sẽ chỉ ra là phần cuối của các headers và phần đầu của body (Đối với STOMP 1.0 thì không có).
* Body (nội dung): chứa nội dung của tin nhắn. Nội dung có thể là chuỗi văn bản hoặc chuỗi nhị phân (binary array). Theo sau của body là một NULL octet, có thể sử dụng ^@ (controll-@ trong ASCII để biểu diễn NULL octet) hoặc nhiều EOLs để chỉ ra là kết thúc body.



Hình 2.19 Cấu trúc của STOMP frame

Trong đó:

* “SEND” là command dùng để xác định hành động là gửi dữ liệu đến server. Theo sau command là một EOL.
* “destination:/queue/a” là header dùng để xác định đích đến, nơi mà message sẽ được gửi đến. Theo sau là một EOL.
* “receipt:message-12345” là header dùng để yêu cầu máy chủ gửi một RECEIPT frame hoặc ERROR frame về cho client sau khi xử lý thành công hoặc thất bại việc gửi message này. Theo sau là một EOL.
* Khoảng trống (một EOL)
* “hello queue a” là body chứa nội dung của message được gửi đi. Theo sau là ^@ đại diện cho một NULL octet chỉ ra việc kết thúc của body.

## Các công nghệ web

### Công nghệ web phía client

#### HTML (Hypertext Markup Language)

HTML (Hypertext Markup Language) là một ngôn ngữ đánh dấu tiêu chuẩn dùng để tạo và cấu trúc các trang web. Nó cung cấp một cách để mô tả cấu trúc của một trang web, bao gồm các phần như tiêu đề, đoạn văn, danh sách, liên kết và media như hình ảnh và video. Một số đặc điểm chính của HTML:

* Sử dụng các thẻ (tags) để mô tả các phần tử của trang web, như <h1>, <p>, <div>, <img>...
* Các thẻ được đặt trong dấu ngoặc < > và thường có cả thẻ mở và thẻ đóng.
* Các phần tử HTML được lồng vào nhau để tạo cấu trúc phân cấp của trang.
* Cho phép định dạng văn bản như in đậm, in nghiêng, gạch chân...
* Cung cấp khả năng tạo liên kết (hyperlinks) giữa các trang web hoặc các phần trong cùng một trang web.
* Cho phép nhúng các phương tiện như hình ảnh, video, âm thanh vào trang web bằng các thẻ tương ứng.

HTML 5 chính thức đưa WebSocket trở thành một phần của tiêu chuẩn web, cung cấp các API để tích hợp và sử dụng WebSocket.



Hình 2.20 Mã HTML

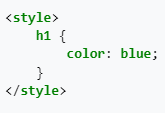
Trong đó:

* “<!DOCTYPE html>” khai báo rằng tài liệu HTML tuân thủ theo tiêu chuẩn HTML 5 giúp trình duyệt biết được cách hiển thị và xử lý trang web đúng cách.
* “<html> </html>” bao quanh toàn bộ nội dung của trang web, cho trình duyệt biết đây là một tài liệu HTML. Một thẻ <html> gồm 2 phần <head> và <body>.
* “<head> </head>” cung cấp các thông tin của trang web như:
  + “<title> </title>”: tiêu đề của trang web.
  + “<meta>”: thông tin về các từ khóa, mô tả…
  + “<link>”: liên kết đến tệp CSS.
* “<body></body>”: chứa nội dung chính của trang web. Chứa các thẻ con khác:
  + “<div> </div>”: tạo ra các khối hoặc vùng chứa nội dung, dùng để tổ chức cấu trúc nội dung của trang web.
  + “<p> </p>”: chứa một đoạn văn bản…

#### CSS (Cascading Style Sheets)

CSS là ngôn ngữ định kiểu để mô tả cách hiển thị các phần tử HTML trên một trang web. Nó được sử dụng để kiểm soát các thuộc tính như font, màu sắc, kích thước, vị trí, và các hiệu ứng khác của các phần tử. Một số đặc điểm chính của CSS:

* Tách biệt nội dung (HTML) khỏi định dạng và trình bày.
* Cho phép định nghĩa các quy tắc định kiểu một lần, sau đó tái sử dụng chúng trên nhiều phần tử HTML.
* Sử dụng hệ thống phân cấp để áp dụng các quy tắc định kiểu.
* Cung cấp hàng trăm thuộc tính để định kiểu các phần tử, như font, màu sắc, kích thước, vị trí, animation...



Hình 2.21 Mã CSS

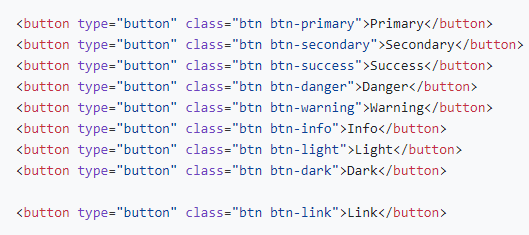
Trong đó:

* “<style> </style>”: định nghĩa các quy tắc CSS trực tiếp trong HTML
  + “color: blue": định nghĩa màu chữ của thẻ <h1>

#### Bootstrap

Bootstrap là một framework CSS mã nguồn mở, được phát triển và duy trì bởi Twitter. Nó cung cấp một tập hợp các công cụ và thành phần giúp thiết kế giao diện web một cách nhanh chóng và đồng nhất. Một số đặc điểm chính của Bootstrap:

* Được thiết kế với triết lý "mobile-first", giúp tạo ra các trang web có thể thích ứng với mọi kích thước thiết bị.
* Cung cấp một thư viện rộng lớn các thành phần giao diện sẵn có, như nút, menu, form, alert, carousel...
* Cho phép tùy biến toàn bộ giao diện, từ màu sắc, font chữ đến kích thước và hiệu ứng.



Hình 2.22 Mã Bootstrap

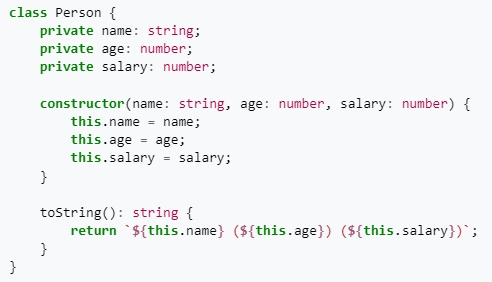
Trong đó:

* “btn” định kiểu button trên trang web
* “btn-\*” cung cấp các biến thể của btn, mỗi biến thể áp dụng một định dạng, màu sắc và hiệu ứng khác nhau.

#### TS (Typescript)

TS là một ngôn ngữ lập trình mở rộng từ JS (Javascript), được phát triển và duy trì bởi Microsoft. Nó cung cấp nhiều tính năng bổ sung so với JavaScript truyền thống, giúp làm cho code JavaScript trở nên mạnh mẽ và dễ bảo trì hơn. Một số đặc điểm chính của TypeScript:

* Bổ sung thêm hệ thống kiểu dữ liệu tĩnh, giúp phát hiện lỗi tại thời gian biên dịch. Các kiểu dữ liệu như number, string, boolean, object, array, etc. có thể được xác định rõ ràng.
* Hỗ trợ các khái niệm lập trình hướng đối tượng như class, interface, inheritance...
* Được biên dịch thành JS tiêu chuẩn, do đó nó hoàn toàn tương thích với các ứng dụng JS hiện có.



Hình 2.23 Mã Typescript

#### Angular

Angular là một framework JavaScript/TypeScript được phát triển và duy trì bởi Google. Nó được sử dụng để xây dựng các ứng dụng web single-page (SPA) và các ứng dụng di động sử dụng công nghệ web. Một số đặc điểm nổi bật của Angular:

* Sử dụng kiến trúc component-based, cho phép chia ứng dụng thành các thành phần tái sử dụng được. Mỗi component quản lý giao diện, logic và dữ liệu của một phần trong ứng dụng.
* Được xây dựng hoàn toàn bằng TypeScript, tận dụng được các tính năng mạnh mẽ của ngôn ngữ này.
* Có hệ thống Dependency Injection (DI) mạnh mẽ, giúp quản lý các phụ thuộc giữa các component và dịch vụ.
* Cung cấp một router mạnh mẽ để điều hướng giữa các view trong ứng dụng SPA.
* Tích hợp sâu với RxJS, một thư viện ReactiveX để xử lý luồng dữ liệu bất đồng bộ.



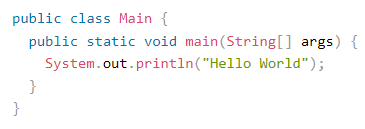
Hình 2.24 Mã Angular

### Công nghệ web phía server

#### Java

Java là một ngôn ngữ lập trình đa mục đích, hướng đối tượng, được phát triển bởi Sun Microsystems (hiện thuộc về Oracle). Java nổi tiếng với tính đa nền tảng, an toàn và hiệu suất cao. Một số đặc điểm chính của Java:

* Là một ngôn ngữ lập trình hướng đối tượng, cung cấp các khái niệm như class, object, inheritance, polymorphism...
* Có thể chạy trên nhiều hệ điều hành khác nhau như Windows, macOS, Linux, etc. mà không cần thay đổi code.
* Cung cấp nhiều tính năng an toàn như quản lý bộ nhớ tự động, kiểm soát truy cập, xử lý ngoại lệ...
* Được thiết kế để đạt hiệu suất cao, đặc biệt với các thuật toán và xử lý dữ liệu phức tạp. Nhờ JIT (Just-In-Time) compiler, Java có thể biên dịch và tối ưu hóa code trong quá trình chạy.

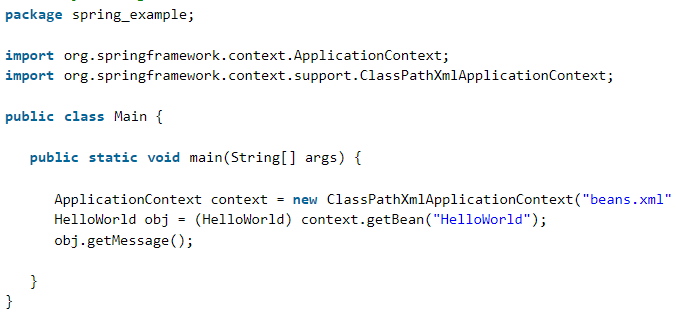


Hình 2.25 Mã Java

#### Spring

Spring là một framework mã nguồn mở cho Java, được phát triển bởi Pivotal Software. Spring cung cấp một nền tảng toàn diện để xây dựng các ứng dụng Java, giúp đơn giản hóa và tiêu chuẩn hóa các nhiệm vụ lập trình thường gặp. Một số đặc điểm chính của Spring:

* Sử dụng mẫu thiết kế DI để quản lý sự phụ thuộc giữa các thành phần của ứng dụng.
* Hỗ trợ lập trình AOP, cho phép tách biệt các concerns (như logging, security, transaction) ra khỏi code chính.
* Cung cấp một mô hình MVC mạnh mẽ để xây dựng các ứng dụng web. Nó hỗ trợ các tính năng như controller, view, model, routing...
* Cung cấp một hệ thống quản lý giao dịch mạnh mẽ, cho phép dễ dàng thực hiện các giao dịch ACID.



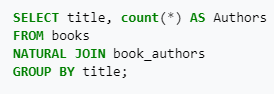
Hình 2.26 Mã Spring

## Cơ sỡ dữ liệu

### SQL (Structured Query Language)

SQL là một ngôn ngữ lập trình được sử dụng để quản lý CSDL quan hệ (RDBMS - Relational Database Management System). SQL được sử dụng để tạo, truy vấn, cập nhật và quản lý dữ liệu trong các hệ thống cơ sở dữ liệu quan hệ như MySQL, PostgreSQL, Oracle, SQL Server, và nhiều hệ thống khác. Các chức năng chính:

* Truy vấn dữ liệu (SELECT): SQL cho phép truy vấn dữ liệu từ các bảng trong CSDL sử dụng câu lệnh SELECT. Bạn có thể chỉ định các điều kiện, sắp xếp, và chọn các cột cụ thể để trả về kết quả mong muốn.
* Cập nhật dữ liệu (INSERT, UPDATE, DELETE): SQL cung cấp các câu lệnh để thêm mới dữ liệu vào bảng (INSERT), cập nhật dữ liệu đã tồn tại (UPDATE), và xóa dữ liệu (DELETE) khỏi bảng.
* Tạo và quản lý cấu trúc dữ liệu (CREATE, ALTER, DROP): SQL cho phép bạn tạo mới các bảng (CREATE TABLE), chỉnh sửa cấu trúc của bảng (ALTER TABLE), và xóa bỏ các bảng hoặc đối tượng khác (DROP TABLE, DROP DATABASE).
* Quản lý dữ liệu (GRANT, REVOKE): SQL cung cấp các câu lệnh để quản lý quyền truy cập và bảo mật dữ liệu, cho phép bạn cấp quyền (GRANT) và thu hồi quyền (REVOKE) cho người dùng và vai trò trong hệ thống cơ sở dữ liệu.



Hình 2.27 Mã SQL

### MySQL

MySQL là một hệ quản trị cơ sở dữ liệu (DBMS) mã nguồn mở, được phát triển và duy trì bởi Oracle Corporation. MySQL là một trong những hệ quản trị cơ sở dữ liệu phổ biến nhất trên thế giới. Một số đặc điểm chính của MySQL:

* Là một phần mềm mã nguồn mở, do đó được cộng đồng phát triển và hỗ trợ rộng rãi.
* Có khả năng mở rộng tốt, có thể xử lý lượng dữ liệu lớn và số lượng truy cập cao.
* Có thể chạy trên nhiều hệ điều hành khác nhau như Windows, Linux, macOS...
* Cung cấp các tính năng bảo mật mạnh mẽ như quản lý quyền truy cập, mã hóa...



Hình 2.28 Icon MySQL

## ORM (Object-Relational Mapping)

### Khái niệm ORM

ORM là một kỹ thuật lập trình trong đó các đối tượng trong ứng dụng được ánh xạ sang các bảng trong cơ sở dữ liệu quan hệ. Nói cách khác, ORM là một cầu nối giữa các đối tượng trong ứng dụng và các bảng trong cơ sở dữ liệu. ORM giải quyết một số vấn đề sau:

* Object–relational impedance mismatch: Trong lập trình hướng đối tượng, các đối tượng được biểu diễn bằng các lớp có thuộc tính và phương thức riêng. Trong khi đó, cơ sở dữ liệu quan hệ sử dụng các bảng có các hàng và cột. Sự không tương thích này gây ra khó khăn trong việc ánh xạ các đối tượng sang cấu trúc dữ liệu của cơ sở dữ liệu.
* Cần phải viết mã nhiều và phức tạp: khi phải thao tác với cả đối tượng và cơ sở dữ liệu, lập trình viên thường phải viết mã nhiều và phức tạp để thực hiện việc ánh xạ dữ liệu giữa hai môi trường này.
* Tăng cường hiệu suất: ORM cung cấp các cơ chế tối ưu hóa truy cập dữ liệu để tăng hiệu suất của ứng dụng. Điều này có thể bao gồm tối ưu hóa câu truy vấn SQL, quản lý kết nối và giao tiếp với cơ sở dữ liệu.
* Giảm sự phụ thuộc vào ngôn ngữ cụ thể: ORM cho phép lập trình viên tạo ra mã không phụ thuộc vào ngôn ngữ cụ thể của cơ sở dữ liệu. Điều này giúp dễ dàng chuyển đổi giữa các hệ quản trị cơ sở dữ liệu khác nhau mà không cần phải thay đổi mã lớn.
* Giảm độ phức tạp của mã: ORM cung cấp các cấu trúc và công cụ để quản lý việc ánh xạ dữ liệu, giúp giảm độ phức tạp của mã và tăng tính maintainable của hệ thống.

Tuy ORM có nhiều ưu điểm, nhưng vẫn tồn tại một số khuyết điểm như:

* Tính hiệu suất có thể giảm khi sử dụng ORM so với việc truy cập trực tiếp vào cơ sở dữ liệu. ORM thường cần thực hiện nhiều phép ánh xạ (mapping) giữa các đối tượng và bản ghi trong cơ sở dữ liệu, điều này có thể tạo ra overhead và làm giảm hiệu suất của ứng dụng.
* ORM thường yêu cầu một số tài nguyên hệ thống để hoạt động, bao gồm tài nguyên bộ nhớ và tài nguyên xử lý. Điều này có thể ảnh hưởng đến khả năng mở rộng và quản lý tài nguyên của hệ thống.
* Sử dụng ORM có thể yêu cầu thời gian và kiến thức để hiểu và triển khai đúng cách. Người phát triển cần phải học và hiểu về cách thức hoạt động của ORM cũng như cách thiết lập và tối ưu hóa cho nhu cầu cụ thể của ứng dụng.

### Lý do ra đời

Một trong những lý do chủ yếu cho sự ra đời của kỹ thuật ORM là giải quyết vấn đề object-relational impedance mismatch - Sự không tương thích trở kháng đối tượng-quan hệ. Object–relational impedance mismatch là sự không tương thích giữa mô hình dữ liệu đối tượng (OODM- object-oriented data models) và hệ quản trị cơ sở dữ liệu quan hệ (RDBMS - Relational Database Management Systems). RDBMS biểu thị dữ liệu ở định dạng bảng, trong khi OODM, biểu diễn nó dưới dạng tập hợp các đối tượng được kết nối với nhau. Việc đọc và ghi các đối tượng bằng RDBMS gặp phải các mismatches:

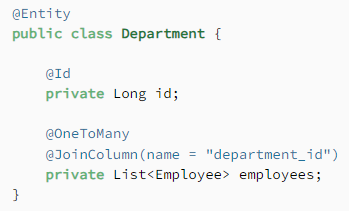
* Granularity (độ chi tiết): Đôi khi OODM có nhiều lớp hơn số lượng bảng tương ứng trong cơ sở dữ liệu. Ví dụ: trong OODM có thuộc tính là enum để biểu diễn hằng số, trong khi RDBMS thường lưu trữ các giá trị đó trong một cột.
* Subtypes / Inheritance (kiểu con / kế thừa): RDBMS không định nghĩa bất cứ điều gì tương tự về tổng thể (mặc dù một số RDBMS có hỗ trợ subtype nhưng không được chuẩn hóa).
* Identity (định danh): RDBMS xác định chính xác danh tính thông qua khóa chính. Tuy nhiên, OODM sử dụng toán tử như == để so sánh các tham chiếu.
* Associations (liên kết): Các liên kết được biểu diễn dưới dạng tham chiếu một chiều hoặc nhiều chiều trong OODM. Trong khi RDBMS sử dụng khái niệm khóa ngoại để truy xuất dữ liệu.
* Data navigation (điều hướng dữ liệu): Trong OODM việc truy xuất dữ liệu bằng cách mạng lưới các đối tượng đi qua liên kết giữa các đối tượng. Ví dụ từ đối tượng A truy cập đến đối tượng B nằm trong A, rồi tiếp tục đến C nằm trong B…Trong khi đó, CSDL sử dụng JOIN để truy vấn nhiều bảng cùng lúc, lấy dữ liệu từ nhiều bảng liên quan và trả về.

## JPA (Jakarta/Java Persistence API)

JPA là một đặc tả kỹ thuật trong Java EE (Java Enterprise Edition) cho phép các nhà phát triển quản lý dữ liệu một cách hiệu quả và độc lập với CSDL. Nó cung cấp một lớp trừu tượng giữa mã ứng dụng và CSDL, cho phép thao tác với dữ liệu bằng các đối tượng Java đơn giản. Các tính năng:

* ORM: JPA ánh xạ các đối tượng Java sang các bảng và cột CSDL. Điều này giúp thao tác với dữ liệu bằng các đối tượng Java đơn giản thay vì viết các truy vấn SQL phức tạp.
* Persistence Context: JPA quản lý một ngữ cảnh liên tục lưu trữ các đối tượng Java được quản lý. Điều này cho phép theo dõi trạng thái của các đối tượng và tự động đồng bộ hóa chúng với cơ sở dữ liệu.
* Entity Manager: JPA cung cấp một lớp trừu tượng gọi là Entity Manager để thao tác với dữ liệu. Entity Manager cho phép thực hiện các thao tác CRUD (Create, Read, Update, Delete) trên các đối tượng Java.
* JPQL (Java Persistence Query Language): JPA cung cấp một ngôn ngữ truy vấn hướng đối tượng gọi là JPQL. JPQL cho phép viết các truy vấn bằng các đối tượng Java thay vì SQL.
* Metadata: JPA sử dụng metadata để ánh xạ các đối tượng Java sang các bảng và cột cơ sở dữ liệu. Metadata có thể được xác định bằng chú thích (annotations) hoặc bằng các tập tin XML.

Ví dụ một số các Metadata của JPA được biểu diễn bằng ạnnotations:



Hình 2.29 Metadata của JPA

Trong đó:

* @Entity đánh dấu lớp Department là một bảng trong CSDL.
* @Id đánh dấu trường id là khóa chính trong bảng.
* @OneToMany đánh dấu Department có mối quan hệ 1 – nhiều với Employee
* @JoinColumn(name = “department\_id”) định nghĩa tên cột được sử dụng khi JOIN với bảng Employee trong CSDL.

Một số triển khai (implementation) của JPA có thể kể đến như: Hibernate, EclipseLink, OpenJPA, Spring Data JPA…

# PHÂN TÍCH VÀ THIẾT KẾ

## Khảo sát các ứng dụng có sẵn

Trước khi phân tích và thiết kế “Ứng dụng nhắn tin thời gian thực” thì tôi đã khảo sát các tính năng, giao diện… của một số ứng dụng nhắn tin/trò chuyện phổ biến hiện nay như: Zalo, Whatsapp, Messenger...

Sau khi khảo sát tôi đưa ra một số nhận xét:

* Các ứng dụng đều có thể hoạt động trên nhiều nền tảng như: mobile (di động), web, desktop (ứng dụng máy tính).
* Đối với nền tảng web, họ sử dụng kỹ thuật CSR (Client-Side Rendering) để xây dựng nhằm tăng trải nghiệm người dùng.
* Các ứng dụng có các tính năng cơ bản như: đăng nhập, đăng ký, gửi/nhận tin nhắn văn bản, chia sẽ hình ảnh/file, chat theo nhóm, chat cá nhân, tạo liên hệ …. Đối với việc nhận gửi tin nhắn thì họ dùng các công nghệ nhắn tin thời gian thực với độ trễ thấp để tối ưu trải nghiệm.
* Về giao diện thì đa số đều có một danh sách các cuộc hội thoại bên trái và bên phải là thông tin chi tiết của cuộc hội thoại đã chọn bao gồm: tin nhắn đã nhận, gửi, người nhận, gửi... Chủ đề màu sắc tươi sáng, đơn giản và dễ sử dụng.
* Ngoài ra, các ứng dụng còn có tính năng thông báo khi có tin nhắn mới.

Kết luận:

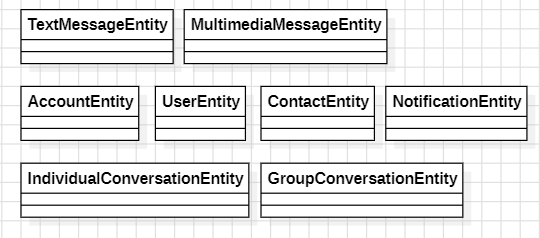
* Để ứng dụng có thể dễ dàng triển khai đa nền tảng thì client và server được phát triển độc lập. Client (nền tảng web) sử dụng công nghệ CSR là Angular để tổ chức và phân bố giao diện thành các thành phần độc lập. Server sẽ xây dựng một tập hợp các APIs cung cấp cho client.
* Đối với việc nhận/gửi tin nhắn thì sử dụng STOMP để định dạng và quản lý các messages trao đổi giữa client và server. Dùng WebSocket để truyền và và nhận các messages hai chiều, liên tục. Đối với tin nhắn dạng văn bản thì gửi và nhận trực tiếp qua WebSocket. Đối với tin nhắn dạng đa phương tiện (hình ảnh, video, file…) thì sử dụng HTTP để upload và dùng WebSocket để trả về liên kết đến các tài nguyên đã lưu trên hệ thống.
* Đối với giao diện, thì sử dụng màu xanh lá cho các nút bấm hay các nội dung cần nổi bật, dùng màu xám làm nền và màu trắng để phân chia rõ các bố cục.
* Đối với việc thông báo khi có tin nhắn mới thì sử dụng STOMP và WebSocket để thông báo người dùng.

## Xác định yêu cầu

* Đăng nhập, đăng xuất.
* Đăng ký, đổi mật khẩu, xóa tài khoản.
* Cập nhật thông tin cá nhân.
* Tạo, sửa, tham gia, rời khỏi cuộc hội thoại nhóm.
* Tạo, xóa liên hệ/danh bạ, khi tạo liên hệ/danh bạ sẽ tự động tạo cuộc hội thoại cá nhân.
* Gửi yêu cầu liên hệ cho người dùng khác.
* Chấp nhận, xóa yêu cầu liên hệ đến.
* Xóa yêu cầu liên hệ đã gửi.
* Nhận gửi tin nhắn. Cho phép gửi tin nhắn dạng văn bản, hình ảnh, video, file…
* Hỗ trợ xem trực tiếp tin nhắn dạng văn bản, file pdf.
* Thu âm và gửi giọng nói.

## Tìm kiếm các đối tượng hệ thống

Đầu tiên là tìm kiếm các đối tượng hệ thống tương ứng với đối tượng thế giới thực cho mô hình. Dựa vào các yêu cầu đã xác định, mã nguồn của một số project trên github và kinh nghiệm của bản thân, tôi đã tìm ra một số danh từ, động từ, cụm động từ. Trong đó, danh từ có trạng thái, hành vi và mối quan hệ với danh từ khác thì có thể trở thành lớp, ngược lại thì trở thành thuộc tính; động từ hoặc cụm động từ có thể trở thành phương thức hoặc nhãn của mối quan hệ. Một số lớp tìm được:



Hình 4.1 Một số lớp tìm được

Trong đó:

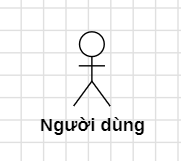
* AccountEntity đại diện cho tài khoản của người dùng.
* UserEntity đại diện cho người dùng.
* TextMessageEntity đại diện cho tin nhắn văn bản.
* MultimediaMessageEntity đại diện cho tin nhắn đa phương tiện.
* ContactEntity đại diện cho liên hệ/danh bạ.
* NotificationEntity đại diện cho thông báo.
* IndividualConversationEntity đại diện cho cuộc hội thoại cá nhân.
* GroupConversationEntity đại diện cho cuộc hội thoại nhóm.

Các lớp sẽ được bổ sung cũng như chi tiết hóa trong các bước tiếp theo.

## Xây dựng sơ đồ trường hợp sử dụng (Use case diagram)

### Tác nhân (Actor)

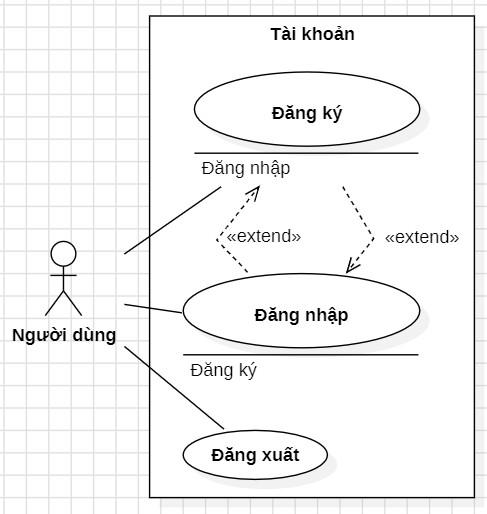
Từ các danh từ đã tìm kiếm được, nhận ra được duy nhất một tác nhân là: người dùng – sử dụng tất cả các tính năng của ứng dụng.



Hình 4.2 Tác nhân người dùng

### Trường hợp sử dụng (Use case)

Từ các động từ và cụm động từ tìm kiếm được, nhận ra các chức năng của ứng dụng (use case), các chức năng được phân chia thành các module nhỏ hơn. Mỗi module chức năng sẽ được thể hiện với một use case diagram và các use case sẽ kèm theo một specification (đặc tả) hay scenario (kịch bản) của chúng. Các module chức năng bao gồm: Tài khoản, Người dùng, Liên hệ, Hội thoại, Hội thoại nhóm.



Hình 4.3 Module Tài khoản

|  |  |
| --- | --- |
| Use case name | Đăng ký |
| Description | Là người dùng, tôi muốn đăng ký tài khoản để sử dụng dịch vụ từ ứng dụng |
| Actor(s) | Người dùng |
| Trigger | Người dùng muốn đăng ký tài khoản để vào ứng dụng |
| Pre-condition(s) | Tài khoản người dùng chưa tồn tại |
| Post-condition(s) | Người dùng đăng ký tài khoản thành công |
| Basic flow | 1) Người dùng nhập tài khoản  2) Người dùng ấn lệnh đăng ký  3) Ứng dụng xác nhận đăng ký thành công và chuyển hướng người dùng đến trang chủ |
| Exception flow | 3b) Ứng dụng xác thực đăng ký không thành công và hiển thị thông báo  Kết thúc use case |

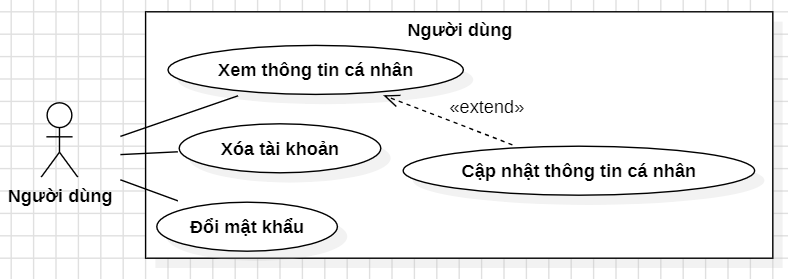
Bảng 4.1 Use case Đăng ký

|  |  |
| --- | --- |
| Use case name | Đăng nhập |
| Description | Là người dùng, tôi muốn đăng nhập vào ứng dụng để sử dụng dịch vụ từ ứng dụng |
| Actor(s) | Người dùng |
| Trigger | Người dùng chọn lệnh đăng nhập |
| Pre-condition(s) | Tài khoản người dùng đã được tạo sẵn |
| Post-condition(s) | Người dùng đăng nhập vào ứng dụng thành công |
| Basic flow | 1) Người dùng nhập tài khoản  2) Người dùng ấn lệnh đăng nhập  3) Ứng dụng xác nhận đăng nhập thành công và chuyển hướng người dùng đến trang chủ |
| Exception flow | 3a) Ứng dụng xác nhận đăng nhập không thành công và hiển thị thông báo  Kết thúc use case |

Bảng 4.2 Use case Đăng nhập

|  |  |
| --- | --- |
| Use case name | Đăng xuất |
| Description | Là người dùng, tôi muốn đăng xuất tài khoản khỏi ứng dụng |
| Actor(s) | Người dùng |
| Trigger | Người dùng ấn lệnh đăng xuất |
| Pre-condition(s) | Người dùng đã đăng nhập vào ứng dụng |
| Post-condition(s) | Người dùng đăng xuất tài khoản thành công |
| Basic flow | 1) Người dùng ấn lệnh đăng xuất  2) Ứng dụng xác nhận đăng xuất thành công  3) Ứng dụng chuyển hướng người dùng đến trang chủ |
| Exception flow | 2a) Ứng dụng xác nhận đăng xuất không thành công  Kết thúc use case |

Bảng 4.3 Use case Đăng xuất



Hình 4.4 Module Người dùng

|  |  |
| --- | --- |
| Use case name | Xem thông tin cá nhân |
| Description | Là người dùng, tôi muốn xem thông tin cá nhân của tôi |
| Actor(s) | Người dùng |
| Trigger | Người dùng ấn lệnh xem thông tin cá nhân |
| Post-condition(s) | Thông tin người dùng được hiển thị trên giao diện |
| Basic flow | 1) Người dùng ấn lệnh xem thông tin  2) Ứng dụng lấy thông tin người dùng thành công và hiển thị lên màn hình |
| Exception flow | 2a) Ứng dụng lấy thông tin người dùng không thành công và hiển thị thông báo  Kết thúc use case |

Bảng 4.4 Use case Xem thông tin cá nhân

|  |  |
| --- | --- |
| Use case name | Cập nhật thông tin cá nhân |
| Description | Là người dùng, tôi muốn cập nhật thông tin cá nhân |
| Actor(s) | Người dùng |
| Trigger | Người dùng ấn lệnh cập nhật thông tin cá nhân |
| Pre-condition(s) | Người dùng đã ấn lệnh xem thông tin cá nhân |
| Post-condition(s) | Thông tin người dùng được cập nhật thành công |
| Basic flow | 1) Người dùng nhập thông tin cá nhân mới và chọn ảnh đại diện  2) Người dùng ấn lệnh xác nhận cập nhật  3) Ứng dụng xác nhận người dùng cập nhật thông tin cá nhân thành công  4) Ứng dụng đóng giao diện cập nhật thông tin cá nhân |
| Exception flow | 2a) Người dùng ấn lệnh hủy  Thực hiện use case 4  Kết thúc use case  3b) Ứng dụng xác nhận người dùng cập nhật thông tin cá nhân không thành công và hiển thị thông báo  Kết thúc use case |

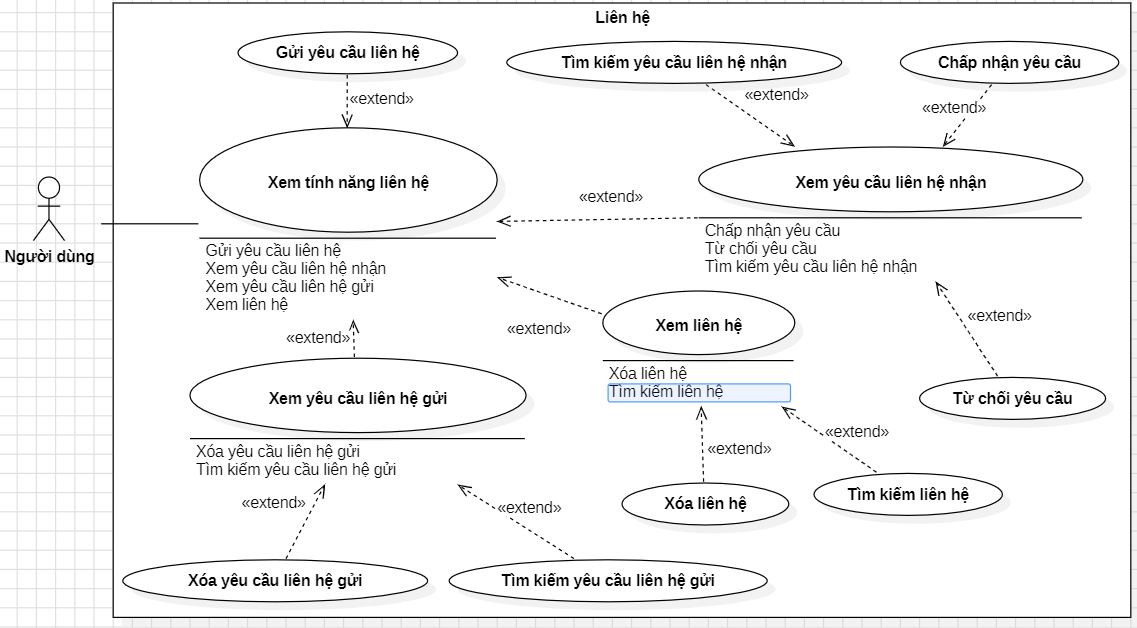
Bảng 4.5 Use case Cập nhật thông tin cá nhân

|  |  |
| --- | --- |
| Use case name | Xóa tài khoản |
| Description | Là người dùng, tôi muốn xóa tài khoản của tôi |
| Actor(s) | Người dùng |
| Trigger | Người dùng ấn lệnh xóa tài khoản |
| Pre-condition(s) | Tài khoản người dùng đã tồn tại |
| Post-condition(s) | Tài khoản người dùng được xóa thành công |
| Basic flow | 1) Người dùng ấn lệnh xóa tài khoản  2) Ứng dụng hiển thị giao diện xác nhận việc xóa tài khoản  3) Người dùng ấn lệnh xác nhận xóa tài khoản  4) Ứng dụng xác nhận người dùng xóa tài khoản thành công  5) Ứng dụng chuyển hướng người dùng đến giao diện đăng nhập |
| Exception flow | 3a) Người dùng ấn lệnh hủy xác nhận xóa tài khoản  3a1) Ứng dụng đóng giao diện xóa tài khoản  Kết thúc use case  4b) Ứng dụng xác nhận xóa tài khoản người dùng không thành công và hiển thị thông báo  Kết thúc use case |

Bảng 4.6 Use case Xóa tài khoản

|  |  |
| --- | --- |
| Use case name | Đổi mật khẩu |
| Description | Là người dùng, tôi muốn đổi mật khẩu tài khoản |
| Actor(s) | Người dùng |
| Trigger | Người dùng muốn đổi mật khẩu tài khoản |
| Pre-condition(s) | - Tài khoản người dùng đã tồn tại  - Người dùng nhập mật khẩu cũ phải chính xác  - Người dùng nhập mật khâu mới phải khác với mật khẩu cũ |
| Post-condition(s) | - Mật khẩu tài khoản của người dùng được đổi thành công |
| Basic flow | 1) Người dùng ấn lệnh đổi mật khẩu  2) Ứng dụng xác nhận người dùng đổi mật khẩu thành công  3) Ứng dụng đóng hộp thoại đổi mật khẩu |
| Exception flow | 2a) Ứng dụng xác nhận người dùng đổi mật khẩu không thành công và hiển thị thông báo.  Kết thúc use case |

Bảng 4.7 Use case Đổi mật khẩu



Hình 4.5 Module Liên hệ

|  |  |
| --- | --- |
| Use case name | Xem tính năng liên hệ |
| Description | Là người dùng, tôi muốn xem các chức năng của liên hệ |
| Actor(s) | Người dùng |
| Trigger | Người dùng ấn lệnh xem chức năng của liên hệ |
| Post-condition(s) | Các chức năng liên hệ được hiển thị trên giao diện |
| Basic flow | 1) Người dùng ấn lệnh xem các chức năng liên hệ  2) Hệ thống hiển thị giao diện các chức năng liên hệ |

Bảng 4.8 Use case Xem tính năng liên hệ

|  |  |
| --- | --- |
| Use case name | Gửi yêu cầu liên hệ |
| Description | Là người dùng, tôi muốn gửi yêu cầu liên hệ cho người dùng khác |
| Actor(s) | Người dùng |
| Trigger | Người dùng ấn lênh gửi yêu cầu liên hệ cho người dùng khác |
| Pre-condition(s) | - Người nhận yêu cầu liên hệ phải tồn tại  - Người nhận yêu cầu liên hệ phải khác với người dùng hiện tại |
| Post-condition(s) | Gửi yêu cầu liên hệ cho người dùng khác thành công |
| Basic flow | 1) Người dùng ấn lệnh mở giao diện gửi yêu cầu liên hệ  2) Ứng dụng hiển thị giao diện thêm liên hệ lên màn hình  3) Người dùng nhập thông tin người dùng cần gửi và ấn lệnh tìm  4) Hệ thống lấy thông tin người dùng cần gửi thành công và hiển thị thông tin người đó trên giao diện  5) Người dùng ấn lệnh gửi yêu cầu  6) Ứng dụng xác nhận gửi yêu cầu thành công  7) Ứng dụng đóng giao diện gửi yêu cầu liên hệ |
| Exception flow | 4a) Hệ thống lấy thông tin người dùng cần gửi không thành công và hiện thị thông báo  Kết thúc use case  6b) Ứng dụng xác nhận gửi yêu cầu không thành công và hiển thị thông báo  Kết thúc use case |

Bảng 4.9 Use case Gửi yêu cầu liên hệ

|  |  |
| --- | --- |
| Use case name | Xem yêu cầu liên hệ đến |
| Description | Là người dùng, tôi muốn xem các yêu cầu liên hệ đã nhận |
| Actor(s) | Người dùng |
| Trigger | Người dùng ấn lệnh xem các yêu cầu liên hệ đã nhận |
| Post-condition(s) | Các yêu cầu liên hệ đã nhận được hiển thị trên giao diện |
| Basic flow | 1) Từ giao diện xem các chức năng liên hệ, người dùng ấn lệnh xem yêu cầu liên hệ đã nhận  2) Ứng dụng lấy dữ liệu thành công và hiển thị trên giao diện |

Bảng 4.10 Use case Xem yêu cầu liên hệ đến

|  |  |
| --- | --- |
| Use case name | Tìm kiếm yêu cầu liên hệ đến |
| Description | Là người dùng, tôi muốn tìm kiếm các yêu cầu liên hệ đã nhận |
| Actor(s) | Người dùng |
| Trigger | Người dùng ấn lệnh tìm kiếm các yêu cầu liên hệ đã nhận |
| Post-condition(s) | Tìm kiếm các yêu cầu liên hệ đã nhận thành công |
| Basic flow | 1) Người dùng nhập từ khóa cần tìm  2) Người dùng ấn lệnh tìm kiếm  3) Ứng dụng xác nhận tìm thành công và hiển thị kết quả tìm kiếm lên màn hình |

Bảng 4.11 Use case Tìm kiếm yêu cầu liên hệ đến

|  |  |
| --- | --- |
| Use case name | Chấp nhận yêu cầu |
| Description | Là người dùng, tôi muốn chấp nhận yêu cầu liên hệ |
| Actor(s) | Người dùng |
| Trigger | Người dùng ấn lệnh chấp nhận yêu cầu liên hệ |
| Pre-condition(s) | Yêu cầu liên hệ chưa tồn tại |
| Post-condition(s) | Chấp nhận thành công yêu cầu liên hệ |
| Basic flow | 1) Người dùng ấn lệnh chấp nhận yêu cầu liên hệ  2) Ứng dụng chấp nhận yêu cầu thành công và xóa yêu cầu liên hệ được chọn |
| Exception flow | 2a) Ứng dụng chấp nhận yêu cầu không thành công và hiển thị thông báo  Kết thúc use case |

Bảng 4.12 Use case Chấp nhận yêu cầu

|  |  |
| --- | --- |
| Use case name | Từ chối yêu cầu |
| Description | Là người dùng, tôi muốn từ chối yêu cầu liên hệ |
| Actor(s) | User |
| Trigger | Người dùng ấn lệnh từ chối yêu cầu liên hệ |
| Pre-condition(s) | Yêu cầu liên hệ đã tồn tại |
| Post-condition(s) | Từ chối thành công yêu cầu liên hệ |
| Basic flow | 1) Từ giao diện xem các yêu cầu liên hệ đã nhận, người dùng ấn lệnh từ chối yêu cầu liên hệ  2) Ứng dụng từ chối yêu cầu thành công và xóa yêu cầu liên hệ được chọn |
| Exception flow | 2a) Ứng dụng từ chối yêu cầu không thành công và hiển thị thông báo  Kết thúc use case |

Bảng 4.13 Use case Từ chối yêu cầu

|  |  |
| --- | --- |
| Use case name | Xem yêu cầu liên hệ nhận |
| Description | Là người dùng, tôi muốn xem các yêu cầu liên hệ đã gửi |
| Actor(s) | Người dùng |
| Trigger | Người dùng ấn lệnh xem các yêu cầu liên hệ đã gửi |
| Post-condition(s) | Các yêu cầu liên hệ đã gửi được hiển thị trên giao diện |
| Basic flow | 1) Người dùng ấn lệnh xem yêu cầu liên hệ đã gửi  2) Ứng dụng lấy dữ liệu thành công và hiển thị trên giao diện |

Bảng 4.14 Use case view Xem yêu cầu liên hệ nhận

|  |  |
| --- | --- |
| Use case name | Tìm kiếm yêu cầu liên hệ gửi |
| Description | Là người dùng, tôi muốn tìm kiếm các yêu cầu liên hệ đã gửi |
| Actor(s) | Người dùng |
| Trigger | Người dùng ấn lệnh tìm kiếm các yêu cầu liên hệ đã gửi |
| Post-condition(s) | Tìm kiếm các yêu cầu liên hệ đã gửi thành công |
| Basic flow | 1) Từ giao diện xem các yêu cầu liên hệ đã gửi, người dùng ấn lệnh tìm kiếm  2) Ứng dụng tìm kiếm thành công và hiển thị kết quả tìm kiếm trên giao diện |

Bảng 4.15 Use case Tìm kiếm yêu cầu liên hệ gửi

|  |  |
| --- | --- |
| Use case name | Xóa yêu cầu liên hệ gửi |
| Description | Là người dùng, tôi muốn xóa các yêu cầu liên hệ đã gửi |
| Actor(s) | Người dùng |
| Trigger | Người dùng ấn lệnh xóa các yêu cầu liên hệ đã gửi |
| Pre-condition(s) | Yêu cầu liên hệ đã gửi đã tồn tại |
| Post-condition(s) | Xóa thành công yêu cầu liên hệ đi |
| Basic flow | 1) Từ giao diện xem các yêu cầu liên hệ đã gửi, người dùng ấn lệnh xóa yêu cầu liên hệ đi  2) Ứng dụng xác nhận xóa thành công và xóa yêu cầu liên hệ đã gửi được chọn trên giao diện |
| Exception flow | 2a) Ứng dụng xác nhận xóa không thành công và hiển thị thông báo  Kết thúc use case |

Bảng 4.16 Use case Xóa yêu cầu liên hệ gửi

|  |  |
| --- | --- |
| Use case name | Xem liên hệ |
| Description | Là người dùng, tôi muốn xem các liên hệ |
| Actor(s) | Người dùng |
| Trigger | Người dùng ấn lệnh xem các liên hệ |
| Post-condition(s) | Các liên hệ được hiển thị trên giao diện |
| Basic flow | 1) Người dùng ấn lệnh xem các liên hệ  2) Ứng dụng lấy dữ liệu thành công và hiển thị trên giao diện |

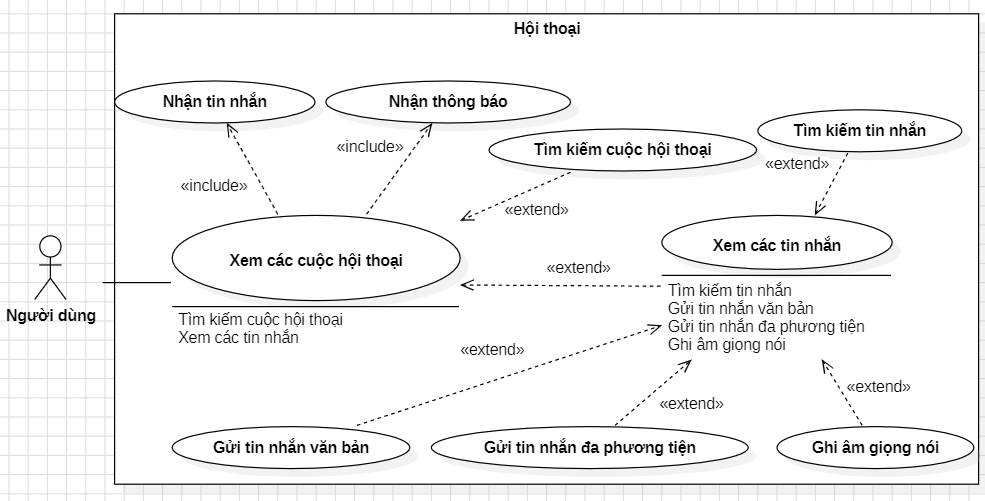
Bảng 4.17 Use case Xem liên hệ

|  |  |
| --- | --- |
| Use case name | Tìm kiếm liên hệ |
| Description | Là người dùng, tôi muốn tìm kiếm các liên hệ |
| Actor(s) | Người dùng |
| Trigger | Người dùng ấn lệnh tìm kiếm các liên hệ |
| Post-condition(s) | Tìm kiếm liên hệ thành công |
| Basic flow | 1) Từ giao diện xem các liên hệ, người dùng ấn lệnh tìm kiếm liên hệ  2) Ứng dụng tìm kiếm liên hệ thành công và hiển thị lên giao diện |

Bảng 4.18 Use case Tìm kiếm liên hệ

|  |  |
| --- | --- |
| Use case name | Xóa liên hệ |
| Description | Là người dùng, tôi muốn xoá liên hệ |
| Actor(s) | User |
| Trigger | Người dùng ấn lệnh xóa liên hệ |
| Pre-condition(s) | Liên hệ đã tồn tại |
| Post-condition(s) | Xóa liên hệ thành công |
| Basic flow | 1) Từ giao diện xem các liên hệ, người dùng ấn lệnh xóa liên hệ  2) Ứng dụng hiển thị giao diện xác nhận xóa  3) Người dùng ấn lệnh xác nhận xóa  4) Ứng dụng xác nhận xóa thành công và xóa liên hệ được chọn ra khỏi giao diện  5) Ứng dụng đóng giao diện xác nhận xóa |
| Exception flow | 3a) Người dùng ấn lệnh hủy  Thực hiện use case 5  Kết thúc use case  4b) Ứng dụng xác nhận xóa không thành công và hiển thị thông báo  Thực hiện use case 5  Kết thúc use case |

Bảng 4.19 Usecase Xóa liên hệ



Hình 4.6 Module Hội thoại

|  |  |
| --- | --- |
| Use case name | Xem các cuộc hội thoại |
| Description | Là người dùng, tôi muốn xem các cuộc hội thoại |
| Trigger | Người dùng đăng nhập thành công vào ứng dụng |
| Post-condition(s) | Các cuộc hội thoại được hiển thị trên màn hình |
| Basic flow | Ứng dụng hiển thị các cuộc hội thoại lên màn hình |

Bảng 4.20 Use case Xem các cuộc hội thoại

|  |  |
| --- | --- |
| Use case name | Nhận tin nhắn |
| Description | Là người dùng, tôi muốn nhận tin nhắn mới từ cuộc hội thoại |
| Trigger | Các cuộc hội thoại hiển thị lên màn hình |
| Post-condition(s) | Tin nhắn mới được hiển thị lên giao diện |
| Basic flow | Ưng dụng hiển thị tin nhắn đó lên giao diện |

Bảng 4.21 Use case Nhận tin nhắn

|  |  |
| --- | --- |
| Use case name | Nhận thông báo |
| Description | Là người dùng, tôi muốn nhận thông báo khi có tin nhắn mới mới từ cuộc hội thoại |
| Trigger | Các cuộc hội thoại hiển thị lên màn hình |
| Post-condition(s) | Thông báo tin nhắn mới được hiển thị lên giao diện |
| Basic flow | Sau khi nhận được tin nhắn mới, ứng dụng hiển thị thông báo tin nhắn mới đó lên giao diện |

Bảng 4.22 Use case Nhận thông báo

|  |  |
| --- | --- |
| Use case name | Tìm kiếm cuộ hội thoại |
| Description | Là người dùng, tôi muốn tìm kiếm các cuộc hội thoại |
| Trigger | Người dùng muốn tìm kiếm các cuộc hội thoại |
| Post-condition(s) | Tìm kiếm cuộc hội thoại thành công |
| Basic flow | 1) N dùng ấn lệnh tìm kiếm cuộc hội thoại  2) Ứng dụng tìm kiếm cuộc hội thoại thành công và hiển thị lên giao diện |

Bảng 4.23 Use case Tìm kiếm cuộ hội thoại

|  |  |
| --- | --- |
| Use case name | Xem các tin nhắn |
| Description | Là người dùng, tôi muốn xem các tin nhắn của cuộc hội thoại đã chọn |
| Trigger | Người dùng chọn cuộc hội thoại từ các cuộc hội thoại |
| Post-condition(s) | Các tin nhắn được hiển thị trên màn hình |
| Basic flow | 1) Từ danh sách các cuộc hội thoại, người dùng chọn một cuộc hội thoại  2) Ứng dụng xác nhận lấy các tin nhắn trong cuộc hội thoại thành công và hiển thị chúng lên màn hình |

Bảng 4.24 Use case Xem các tin nhắn

|  |  |
| --- | --- |
| Use case name | Tìm kiếm tin nhắn |
| Description | Là người dùng, tôi muốn tìm kiếm các tin nhắn cụ thể của cuộc hội thoại |
| Trigger | Người dùng ấn lệnh tìm kiếm |
| Post-condition(s) | Các tin nhắn phù hợp được hiển thị lên màn hình |
| Basic flow | 1) Từ danh sách các cuộc hội thoại, người dùng chọn lệnh tìm kiếm tin nhắn  2) Ứng dụng xác nhận tìm kiếm tin nhắn trong cuộc hội thoại thành công và hiển thị chúng lên màn hình |

Bảng 4.25 Use case Tìm kiếm tin nhắn

|  |  |
| --- | --- |
| Use case name | Gửi tin nhắn văn bản |
| Description | Là người dùng, tôi muốn gửi tin nhắn văn bản |
| Trigger | Người dùng ấn lệnh gửi tin nhắn |
| Pre-condition(s) | Nội dung tin nhắn được gửi không được rỗng |
| Post-condition(s) | Tin nhắn được gửi thành công |
| Basic flow | 1) Người dùng nhập nội dung tin nhắn cần gửi  2) Người dùng ấn lệnh gửi  3) Ứng dụng xác nhận gửi thành công |
| Exception flow | 3a) Ứng dụng xác nhận gửi không thành công và hiển thị thông báo  Kết thúc use case |

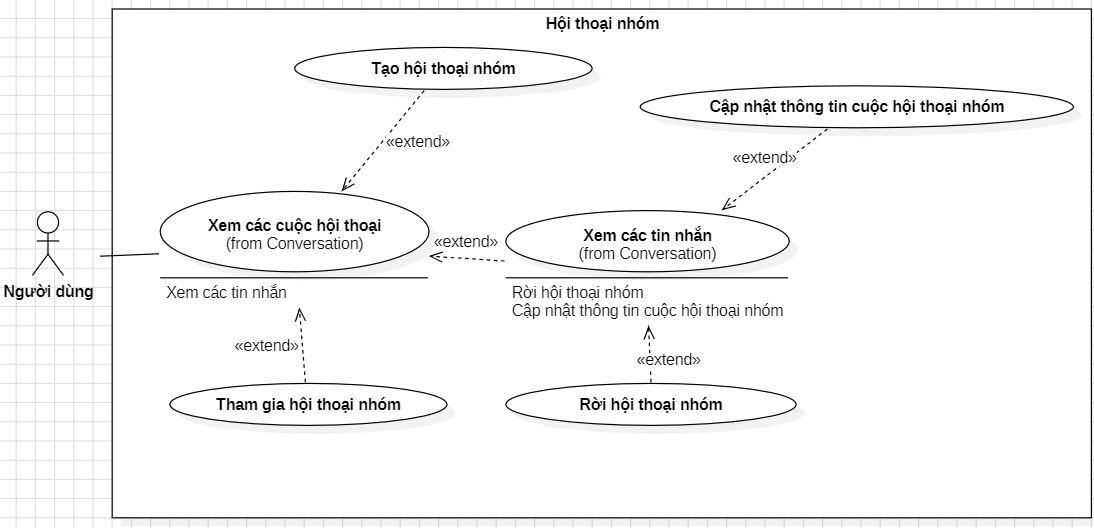
Bảng 4.26 Use case Gửi tin nhắn văn bản

|  |  |
| --- | --- |
| Use case name | Gửi tin nhắn đa phương tiện |
| Description | Là người dùng, tôi muốn gửi tin nhắn đa phương tiện |
| Trigger | Người dùng ấn lệnh gửi tin nhắn |
| Pre-condition(s) | Dữ liệu tin nhắn được gửi không được rỗng |
| Post-condition(s) | Tin nhắn được gửi thành công |
| Basic flow | 1) Người dùng chọn dữ liệu đa phương tiện cần gửi  2) Người dùng ấn lệnh gửi  3) Ứng dụng xác nhận gửi thành công |
| Exception flow | 3a) Ứng dụng xác nhận gửi không thành công và hiển thị thông báo  Kết thúc use case |

Bảng 4.27 Use case Gửi tin nhắn đa phương tiện

|  |  |
| --- | --- |
| Use case name | Ghi âm giọng nói |
| Description | Là người dùng, tôi muốn gửi giọng nói vừa được ghi âm |
| Trigger | Người dùng ấn lệnh gửi |
| Pre-condition(s) | Dữ liệu giọng nói được gửi không được rỗng |
| Post-condition(s) | Dữ liệu được gửi thành công |
| Basic flow | 1) Người dùng ấn lệnh ghi âm  2) Ứng dụng hiển thị giao diện ghi âm  3) Người dùng ấn lệnh bắt đầu ghi âm  4) Người dùng ấn lệnh ngưng  5) Người dùng ấn lệnh gửi  6) Ứng dụng xác nhận gửi thành công |
| Exception flow | 5a) Người dùng ấn lệnh hủy  Kết thúc use case  6b) Ứng dụng xác nhận gửi không thành công và hiển thị thông báo  Kết thúc use case |

Bảng 4.28 Use case Ghi âm giọng nói



Hình 4.7 Module Hội thoại nhóm

|  |  |
| --- | --- |
| Use case name | Tạo hội thoại nhóm |
| Description | Là người dùng, tôi muốn tạo một cuộc hội thoại nhóm |
| Trigger | Người dùng ấn lệnh tạo |
| Pre-condition(s) | Dữ liệu cuộc hội thoại không được rỗng |
| Post-condition(s) | Tạo cuộc hội thoại nhóm thành công |
| Basic flow | 1) Người dùng ấn lệnh tạo cuộc hội thoại nhóm  2) Ứng dụng hiển thị giao diện tạo cuộc hội thoại nhóm  3) Người dùng nhập thông tin cuộc hội thoại nhóm  4) Người dùng ấn lệnh tạo  5) Ứng dụng xác nhận tạo thành công  6) Ứng dụng đóng giao diện tạo cuộc hội thoại nhóm  7) Ứng dụng hiển thị cuộc hội thoại được tạo lên màn hình |
| Exception flow | 4a) Người dùng ấn lệnh hủy  Thực hiện bước 6  Kết thúc use case  5b) Ứng dụng xác nhận tạo không thành công  5b1) Ứng dụng hiển thị thông báo  Kết thúc use case |

Bảng 4.29 Use case Tạo hội thoại nhóm

|  |  |
| --- | --- |
| Use case name | Cập nhật thông tin cuộc hội thoại nhóm |
| Description | Là người dùng, tôi muốn cập nhật thông tin cuộc hội thoại nhóm |
| Trigger | Người dùng ấn lệnh cập nhật |
| Post-condition(s) | Cập nhật thông tin cuộc hội thoại nhóm thành công |
| Basic flow | 1) Người dùng ấn lệnh cập nhật  2) Ứng dụng hiển thị giao diện cập nhật thông tin cuộc hội thoại nhóm  3) Người dùng nhập thông tin mới  4) Người dùng ấn lệnh cập nhật  5) Ứng dụng xác nhận cập nhật thành công  6) Ứng dụng đóng giao diện cập nhật |
| Exception flow | 4a) Người dùng ấn lệnh hủy  Thực hiện bước 6  Kết thúc use case  5b) Ứng dụng xác nhận cập nhật không thành công  5b1) Ứng dụng hiển thị thông báo  Kết thúc use case |

Bảng 4.30 Use case Cập nhật thông tin cuộc hội thoại nhóm

|  |  |
| --- | --- |
| Use case name | Tham gia hội thoại nhóm |
| Description | Là người dùng, tôi muốn tham gia vào cuộc hội thoại nhóm |
| Trigger | Người dùng ấn lệnh tham gia |
| Post-condition(s) | Người dùng tham gia cuộc hội thoại nhóm thành công |
| Basic flow | 1) Người dùng ấn lệnh tham gia  2) Ứng dụng hiển thị giao diện tham gia cuộc hội thoại nhóm  3) Người dùng nhập thông tin nhóm  4) Người dùng ấn lệnh tham gia  5) Ứng dụng xác nhận tham gia thành công  6) Ứng dụng đóng giao diện tham gia  7) Cuộc hội thoại nhóm vừa tham gia hiển thị lên màn hình |
| Exception flow | 4a) Người dùng ấn lệnh hủy  Thực hiện bước 6  Kết thúc use case  5b) Ứng dụng xác nhận tham gia không thành công  5b1) Ứng dụng hiển thị thông báo  Kết thúc use case |

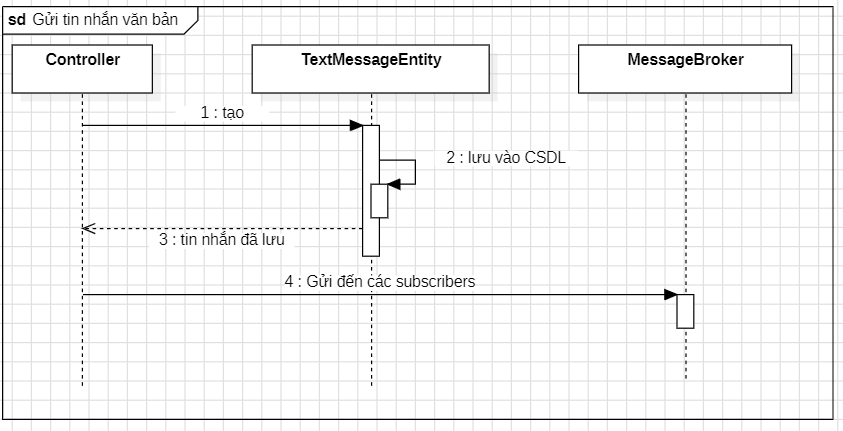
Bảng 4.31 Use case Tham gia hội thoại nhóm

|  |  |
| --- | --- |
| Use case name | Rời hội thoại nhóm |
| Description | Là người dùng, tôi muốn rời khỏi cuộc hội thoại nhóm |
| Trigger | Người dùng ấn lệnh rời khỏi |
| Post-condition(s) | Người dùng rời khỏi cuộc hội thoại nhóm thành công |
| Basic flow | 1) Người dùng ấn lệnh rời khỏi  2) Ứng dụng hiển thị giao diện xác nhận  3) Người dùng ấn lệnh xác nhận  4) Ứng dụng xác nhận rời khỏi thành công  5) Ứng dụng đóng giao diện xác nhận  6) Ứng dụng xóa cuộc hội thoại vừa rời khỏi khỏi màn hình |
| Exception flow | 3a) Người dùng ấn lệnh hủy  Thực hiện bước 5  Kết thúc use case  4b) Ứng dụng xác nhận rời khỏi không thành công  Thực hiện use case 5  4b1) Ứng dụng hiển thị thông báo  Kết thúc use case |

Bảng 4.32 Use case Rời hội thoại nhóm

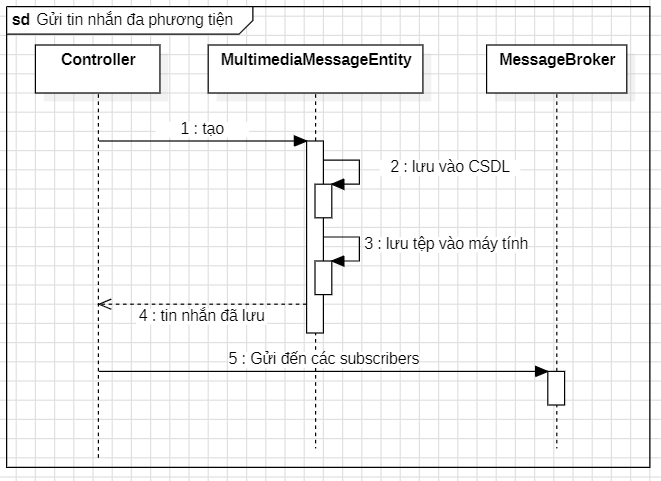
## Xây dựng sơ đồ tuần tự (Sequence diagram)

### Tiến trình gửi tin nhắn văn bản



Hình 4.8 Tiến trình gửi tin nhắn văn bản

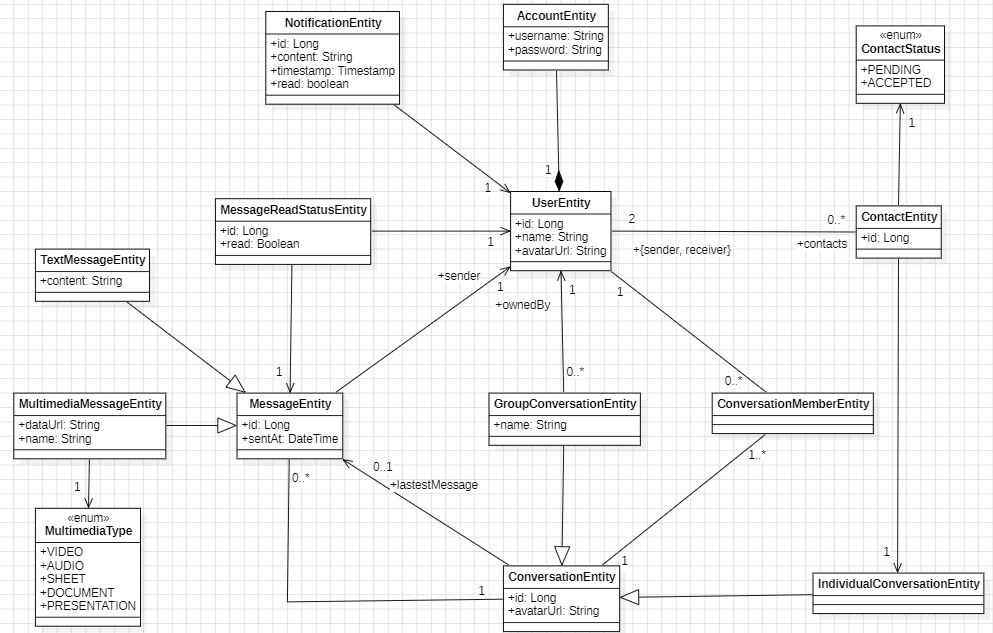
### Tiến trình gửi tin nhắn đa phương tiện



Hình 4.9 Tiến trình gửi tin nhắn đa phương tiện

## Xây dựng sơ đồ lớp (Class diagram)

Sau các bước trên, thì tôi bổ sung các thuộc tính và mối quan hệ cho sơ đồ lớp. Lý do cho việc không thêm phương thức là để đảm bảo nguyên tắc S (Single-responsibility Principle) trong SOLID. Các lớp bên dưới đóng vai trò là Entity (được ánh xạ thành bảng trong CSDL)



Hình 4.10 Sơ đồ lớp

## Xây dựng các kênh giao tiếp trong STOMP

Khi người dùng đăng nhập thành công vào ứng dụng, thì danh sách các cuộc hội thoại sẽ được hiển thị. Khi đó, STOMP client sẽ tiến hành subscribe các kênh dựa theo các id của cuộc hội thoại.

Endpoint dùng để gửi text message:

* “/conversations/{conversationId}/textMessages”

Endpoint dùng để gửi multimedia message:

* “/conversations/{conversationid}/multimediaMessages”

Khi client gửi các message qua các enpoint trên thì server sẽ tự động xử lý và gửi các message đã được xử lý cho các subscribers thông qua kênh mà họ đã subscribe. Kênh để lắng nghe tin nhắn mới của một cuộc hội thoại:

* “/public/conversations/{conversationId}/messages”

## Kiến trúc của ứng dụng

Sau khi có sơ đồ lớp, tôi cần đảm bảo ứng dụng có một kiến trúc tốt nhằm tăng khả năng dễ bảo trì và mở rộng. Với ứng dụng này tôi sử dụng mô hình Controller-Service-Repository.

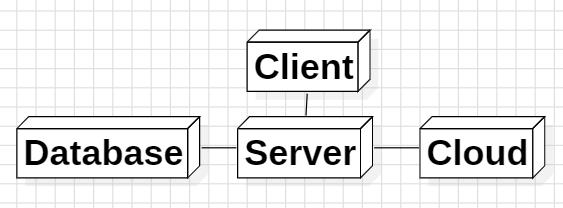
Trong đó:

* Controller chịu trách nhiệm nhận các request từ client và đưa request đó đến service đến tiến hành xử lý, sau đó thì nhận kết quả từ service để gửi response đến client.
* Service chịu trách nhiệm xử lý các dữ liệu mà controller đã đưa, service sẽ xử lý các logic chính của chức năng và truyền dữ liệu sau khi xử lý cho repository để lưu vào CSDL.
* Repository cung cấp một số các abstract method để service có thể sử dụng, repository sẽ nhận các entity từ service để lưu vào CSDL.

Ứng dụng sẽ không dùng các Entity để ánh xạ trực tiếp sang JSON để gửi cho client. Thay vào đó, ứng dụng sẽ dùng các DTO (data stransfer object) có cấu trúc đơn giản, phù hợp hơn với yêu cầu của client để ánh xạ sang JSON.

Ngoài ra, ứng dụng còn bao gồm các lớp khác như: Validator (xác thực các ràng buộc của request), Mapper (ánh xạ Entity sang DTO và ngược lại), FileUtil (lớp tiện ích dùng để xử lý các file trong server)…

## Xây dựng sơ đồ triển khai



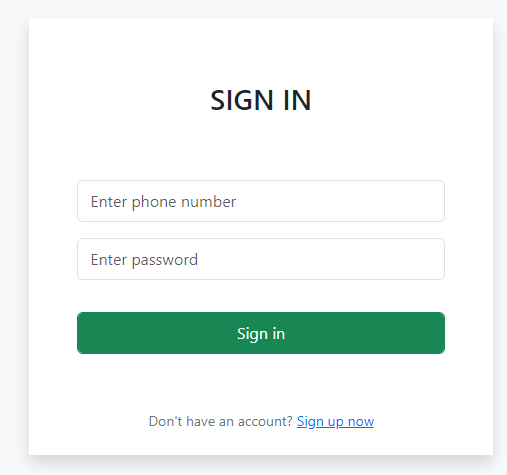
Hình 4.11 Sơ đồ triển khai

Trong đó:

* Client được triển khai trên một host, host này là GithubPages.
* Server được triển khai trên một host, host này là render.
* Database được triển khai trên một host, host này là google cloud sql.
* Cloud thì do bên thứ 3 cung cấp dịch vụ để sử dụng.

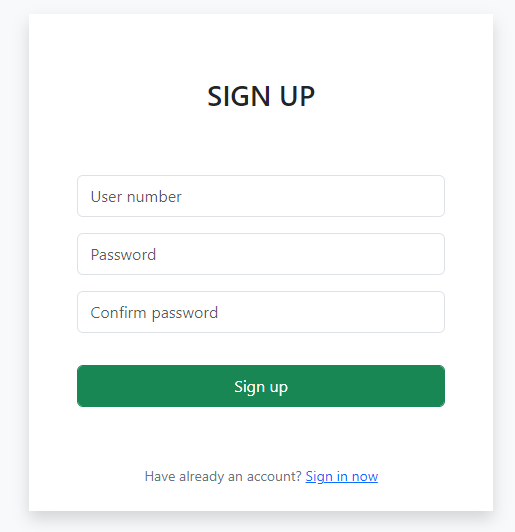
# KẾT QUẢ ĐẠT ĐƯỢC

## Giao diện đăng nhập



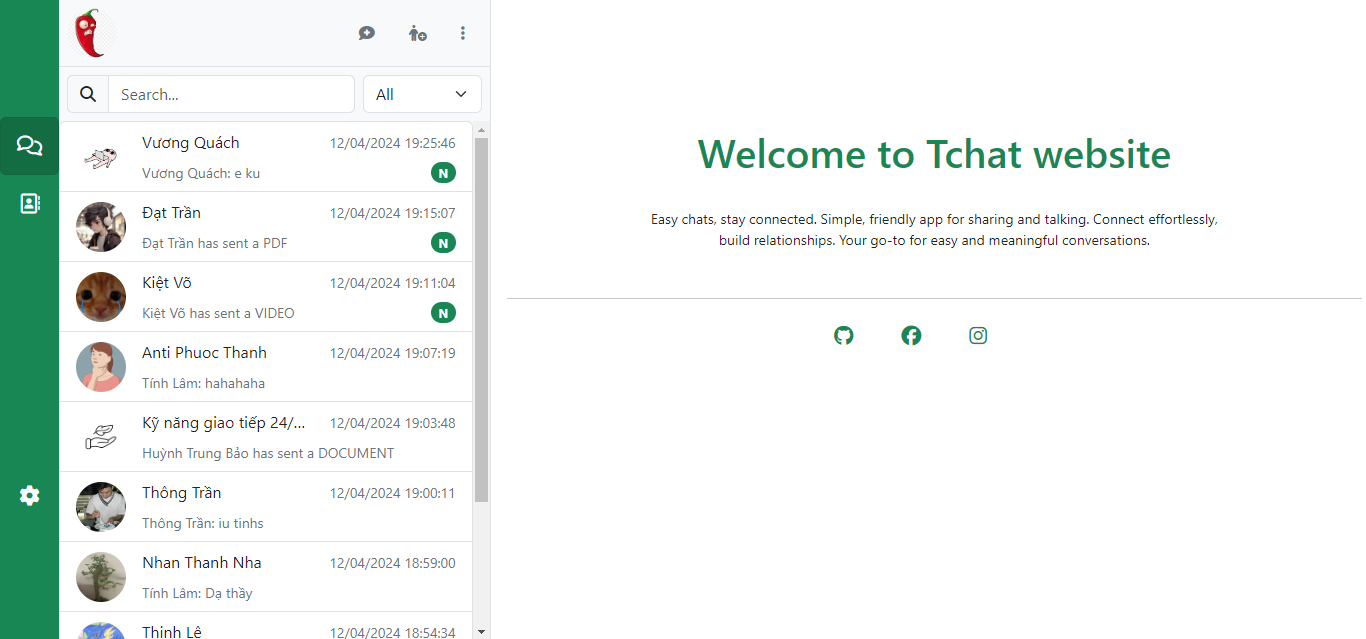
Hình 5.1 Giao diện đăng nhập

## Giao diện đăng ký



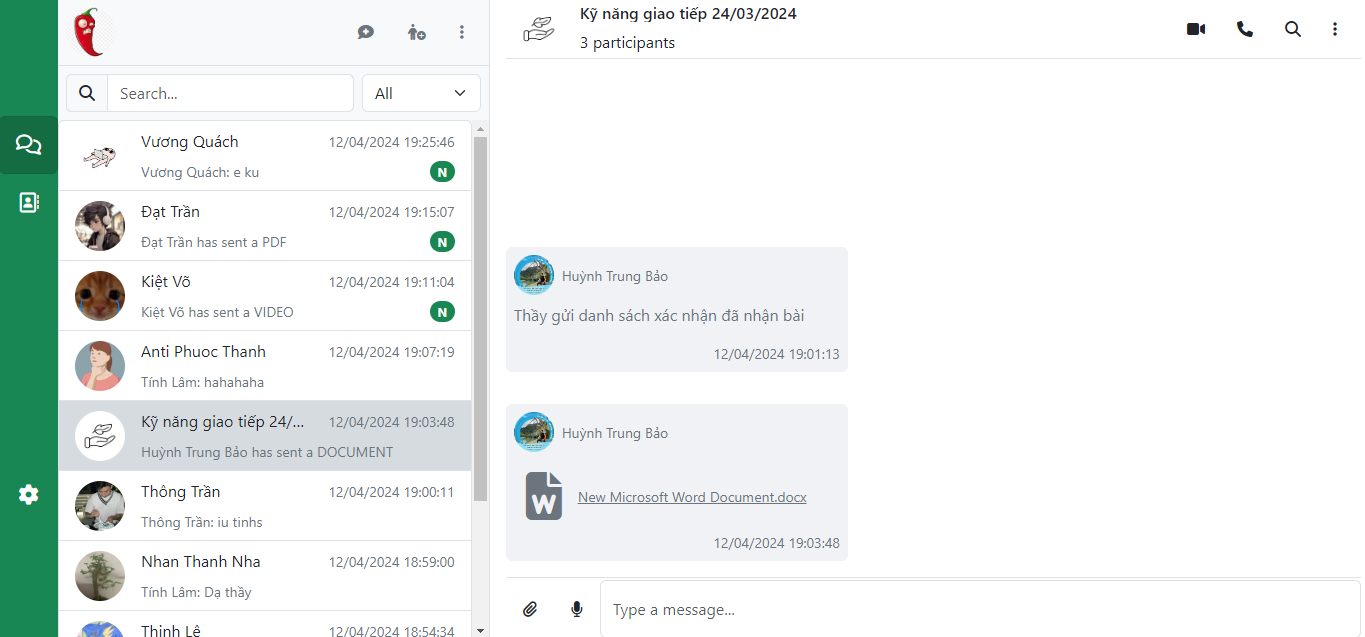
Hình 5.2 Giao diện đăng ký

## Giao diện trang chủ



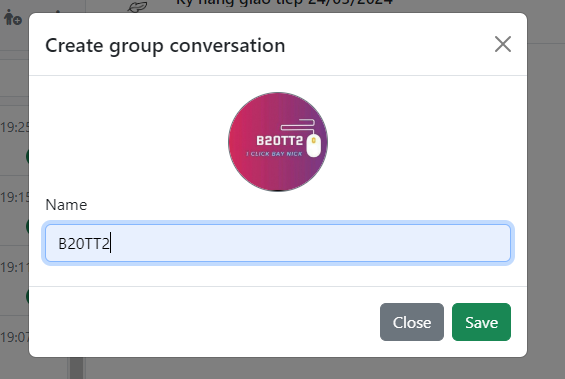
Hình 5.3 Giao diện trang chủ

## Giao diện xem tin nhắn



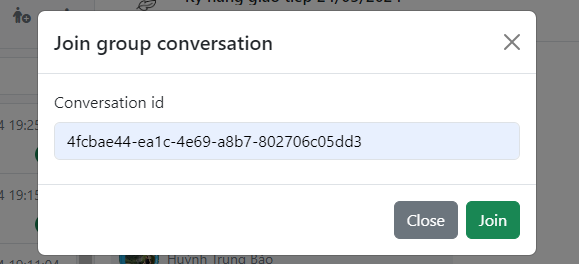
Hình 5.4 Giao diện xem tin nhắn

## Giao diện tạo hội thoại nhóm



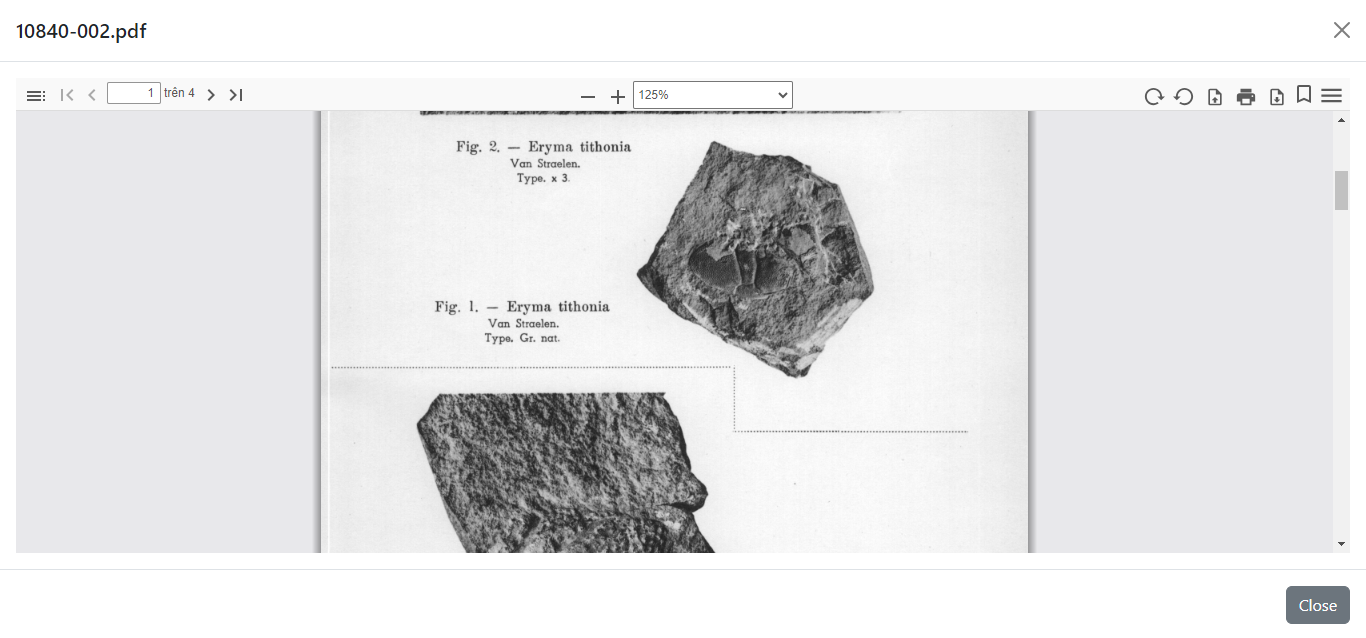
Hình 5.5 Giao diện tạo hội thoại nhóm

## Giao diện tham gia hội thoại nhóm



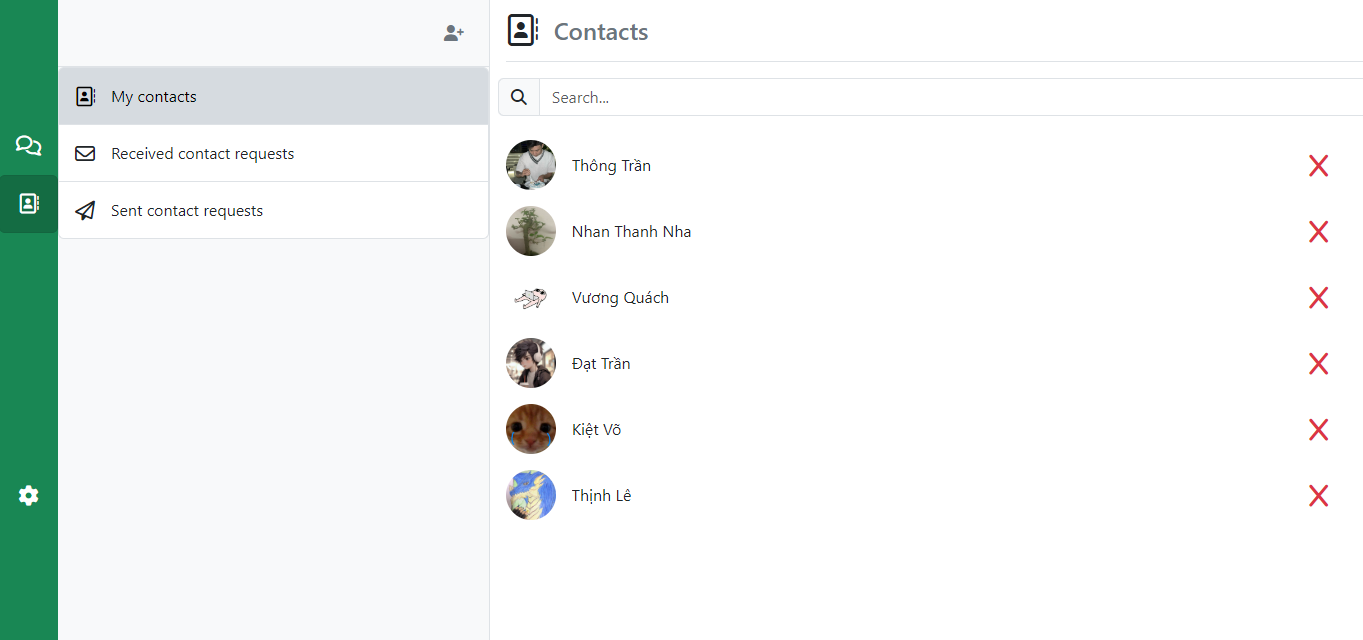
Hình 5.6 Giao diện tham gia hội thoại nhóm

## Giao diện xem trực tiếp file



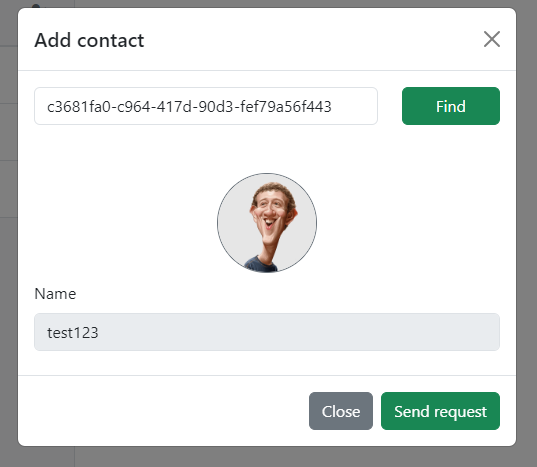
Hình 5.7 Giao diện đọc file

## Giao diện các liên hệ



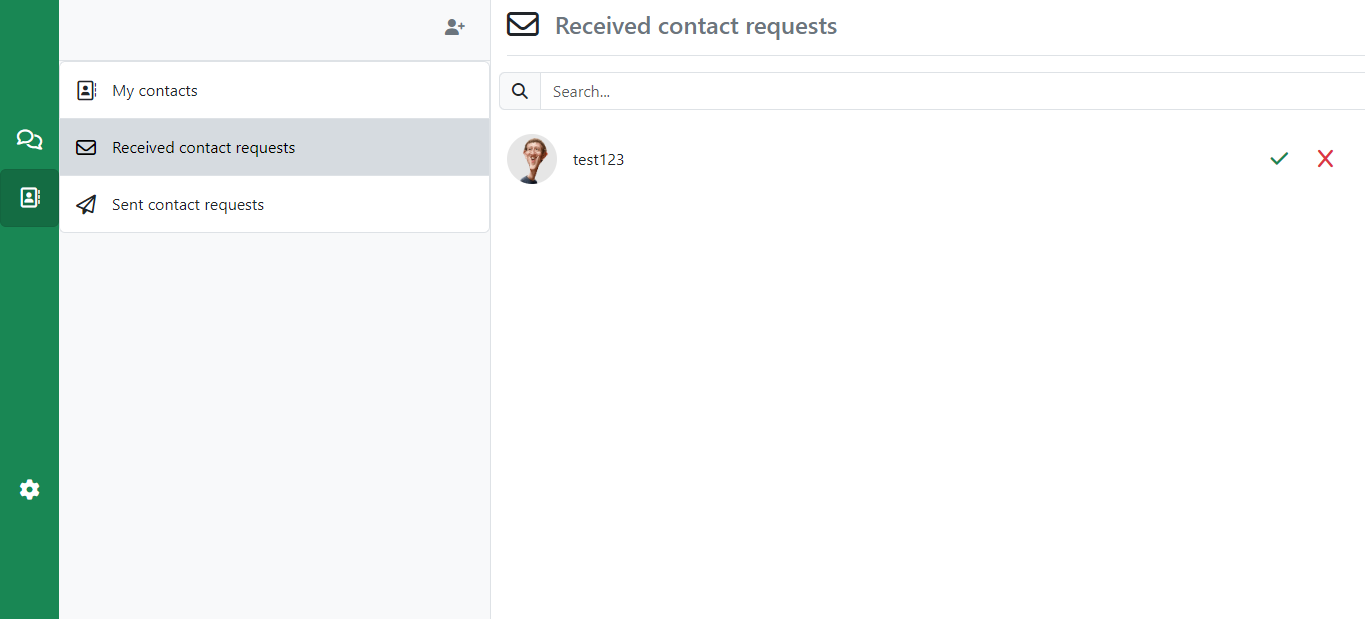
Hình 5.8 Giao diện liên hệ

## Giao diện thêm liên hệ



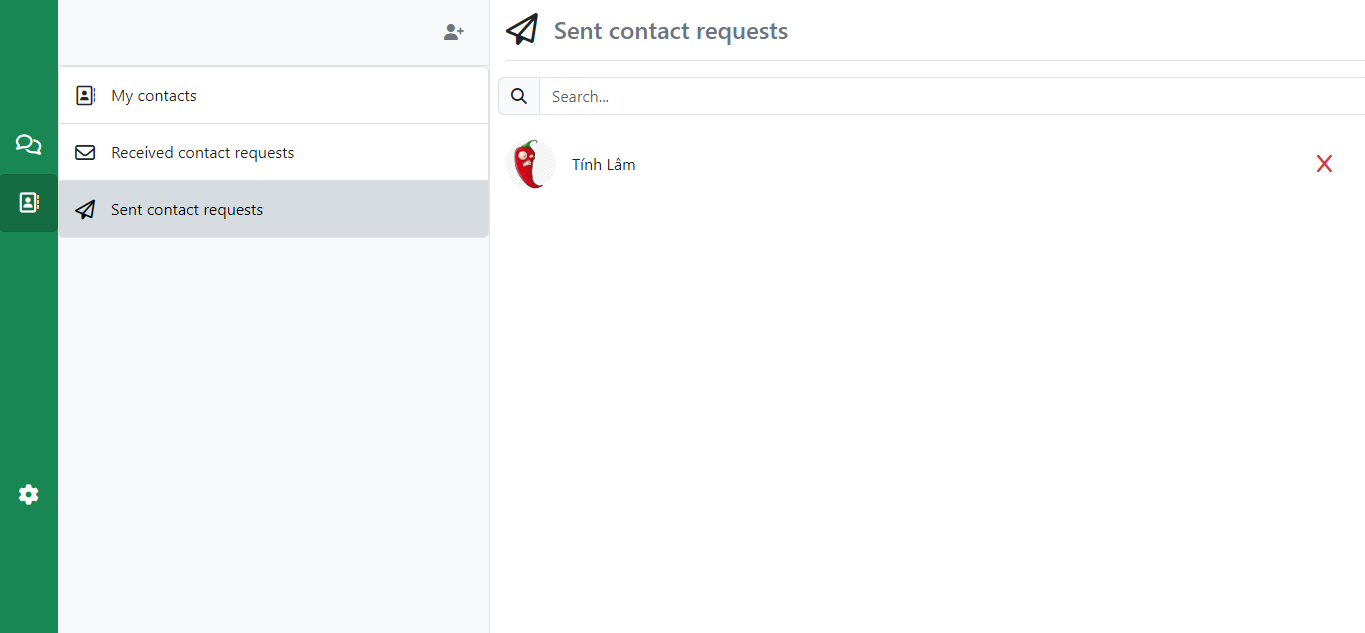
Hình 5.9 Giao diện thêm liên hệ

## Giao diện yêu cầu liên hệ đã nhận



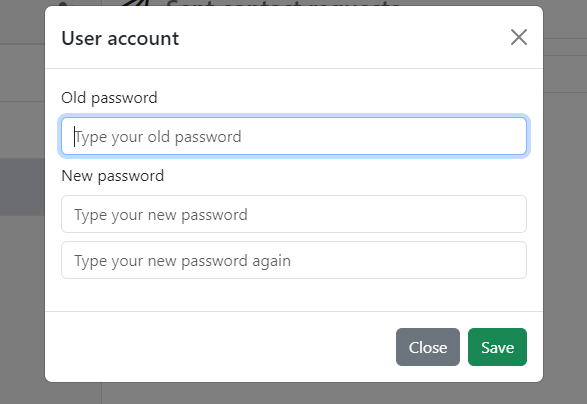
Hình 5.10 Giao diện yêu cầu liên hệ đã nhận

## Giao diện yêu cầu liên hệ đã gửi



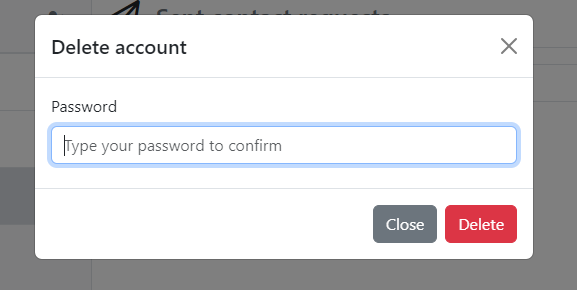
Hình 5.11 Giao diện yêu cầu liên hệ đã gửi

## Giao diện đổi mật khẩu



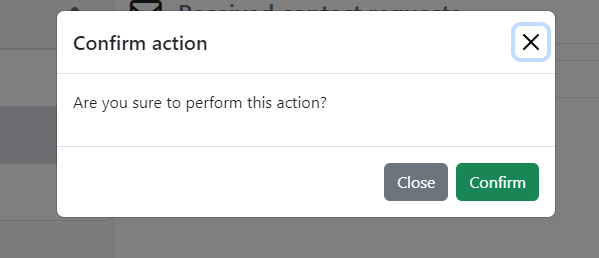
Hình 5.12 Giao diện đổi mật khẩu

## Giao diện xóa tài khoản



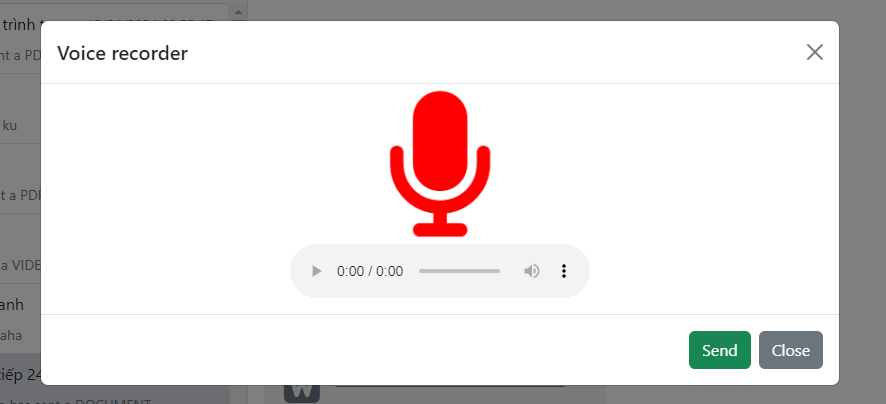
Hình 5.13 Giao diện xóa tài khoản

## Giao diện xác nhận



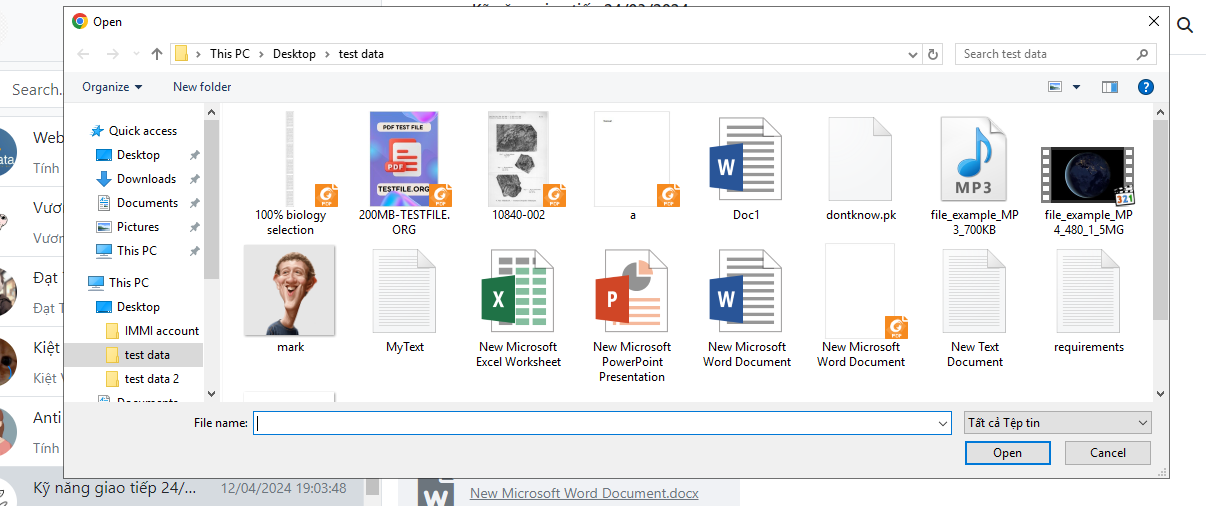
Hình 5.14 Giao diện xác nhận

## Giao diện ghi âm



Hình 5.15 Giao diện ghi âm

## Giao diện chọn file



Hình 5.16 Giao diện chọn file

# KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

## Kết luận

“Ứng dụng nhắn tin thời gian thực” đã giải quyết hầu hết các tính năng cơ bản của một ứng dụng nhắn tin chuyên nghiệp, các tính năng bao gồm:

* Đăng ký tài khoản.
* Xác thực và phân quyền người dùng.
* Quản lý thông tin tài khoản, thông tin người dùng.
* Quản lý các cuộc hội thoại.
* Quản lý các liên hệ, yêu cầu liên hệ đến và yêu cầu liên hệ đi.
* Nhận và gửi tin nhắn, hình ảnh, video, tập tin...
* Gửi thông báo tin nhắn mới.
* Ghi âm giọng nói.

Trong quá trình nghiên cứu, phát triển “Ứng dụng nhắn tin thời gian thực” tôi đã gặp một số thuận lợi và khó khăn:

* Thuận lợi:
  + Với kinh nghiệm sử dụng ngôn ngữ Java và nội dung về OOP thì tôi khá dễ dàng trong việc xây dựng kiến trúc ứng dụng theo ý muốn.
  + Về phần giao diện, tôi đã sử dụng bootstrap framework cho việc xây dựng giao diện vì thế nó cũng không quá phức tạp.
* Khó khăn:
  + Các giao thức mạng quá phức tạp. Việc tìm hiểu về cấu trúc và cách hoạt động của các giao thức này khiến tôi mất rất nhiều thời gian và công sức.
  + Angular và kiến trúc component-based, do tôi chưa từng tiếp xúc với các công nghệ SPA vì thế nên việc tìm hiểu chúng cũng mất nhiều thời gian và công sức.
  + Do hạn chế về mặt thời gian nên chưa xây dựng được đầy đủ các tính năng.

## Hướng phát triển

Trong tương lai, tôi sẽ phát triển cho ứng dụng thêm nhiều tính năng mới cũng như tối ưu hóa hiệu suất và trải nghiệm người dùng.

* Quản lý các yêu cầu tham gia nhóm của quản trị viên.
* Cho phép gửi các emoji, biểu tượng cảm xúc.
* Gọi điện âm thành và video trực tuyến.
* Theo dõi trạng thái đọc của mỗi tin nhắn.
* Cho phép xem trước các file như document, sheet hoặc presentation.
* Sử dụng kỹ thuật streaming để tối ưu cho việc tải video, hình ảnh.

# TÀI LIỆU THAM KHẢO

1. Đặng Văn Đức (2002), *Giáo trình phân tích và thiết kế hướng đối tượng 2002.*
2. Joshua Bloch (2017), Effective Java (3rd Edition).
3. Robert C. Martin (2008), *Clean Code: A Handbook of Agile Software Craftsmanship.*
4. RESTful API. Truy cập ngày 18/04/2024, từ [*https://stackoverflow.blog/2020/03/02/best-practices-for-rest-api-design/*](https://stackoverflow.blog/2020/03/02/best-practices-for-rest-api-design/)
5. STOMP. (2012). Truy cập ngày 18/04/2024, từ[*https://stomp.github.io/*](https://stomp.github.io/)
6. WebSocket. (2011). Truy cập ngày 18/04/2024, từ [*https://en.wikipedia.org/wiki/WebSocket*](https://en.wikipedia.org/wiki/WebSocket)
7. Angular docs. (2016). Truy cập ngày 18/04/2024, từ [*https://angular.io/docs*](https://angular.io/docs)
8. Spring docs. (2002), Truy cập ngày 18/04/2024, từ [*https://docs.spring.io/spring-framework/reference/index.html*](https://docs.spring.io/spring-framework/reference/index.html)
9. Java tutorial. Truy cập ngày 18/04/2024, từ <https://www.tutorialspoint.com/java/index.htm>