ĐẠI HỌC BÁCH KHOA HÀ NỘI

Nghiên cứu tốt nghiệp 2

Thiết kế xây dựng web nhắn tin, gọi điện

LĂNG TRỌNG TIẾN
tien.lt225933@sis.hust.edu.vn

Chương trình đào tạo: Công nghệ thông tin Việt-Nhật

Giảng viên hướng dẫn: TS. Ngô Văn Linh

Khoa: Khoa học máy tính

Trường: Công nghệ thông tin và Truyền thông

HÀ NỘI, 12/2025

ĐẠI HỌC BÁCH KHOA HÀ NỘI

Nghiên cứu tốt nghiệp 2

Thiết kế xây dựng web nhắn tin, gọi điện

LĂNG TRỌNG TIẾN
tien.lt225933@sis.hust.edu.vn

Chương trình đào tạo: Công nghệ thông tin Việt-Nhật

Giảng viên hướng dẫn: TS. Ngô Văn Linh _____

Chữ kí GVHD

Khoa: Khoa học máy tính

Trường: Công nghệ Thông tin và Truyền thông

HÀ NỘI, 12/2025

LỜI CẢM ƠN

Em xin gửi lời cảm ơn chân thành và sâu sắc nhất tới thầy **TS. Ngô Văn Linh**, người đã tận tình hướng dẫn, chỉ bảo và tạo điều kiện tốt nhất để em hoàn thành đề tài "Thiết kế xây dựng web nhắn tin, gọi điện". Những kiến thức và kinh nghiệm quý báu mà Thầy truyền đạt không chỉ giúp em hoàn thành Đồ án Nghiên cứu tốt nghiệp 2 mà còn là hành trang vững chắc cho con đường sự nghiệp sau này.

Em cũng xin gửi lời cảm ơn đến quý Thầy, Cô giáo trường Công nghệ Thông tin và Truyền thông - Đại học Bách khoa Hà Nội đã tận tâm giảng dạy, trang bị cho em những kiến thức nền tảng và chuyên sâu trong suốt quá trình học tập và rèn luyện tại trường.

Cuối cùng, em xin gửi lời tri ân tới gia đình và bạn bè, những người đã luôn bên cạnh động viên, ủng hộ và giúp đỡ em vượt qua những khó khăn trong suốt thời gian qua.

Mặc dù đã có nhiều cố gắng, nhưng do thời gian hạn hẹp và kiến thức còn hạn chế, bản đồ án này khó tránh khỏi những thiếu sót. Em rất mong nhận được sự thông cảm và những ý kiến đóng góp quý báu từ quý Thầy, Cô và các bạn để em có thể rút kinh nghiệm và hoàn thiện bản thân hơn.

Em xin chân thành cảm ơn!

TÓM TẮT NỘI DUNG ĐỒ ÁN

Trong bối cảnh công nghệ thông tin phát triển mạnh mẽ, nhu cầu trao đổi thông tin trực tuyến qua tin nhắn và cuộc gọi video ngày càng tăng cao, đòi hỏi các ứng dụng phải đảm bảo tốc độ nhanh, độ trễ thấp và khả năng xử lý đa phương tiện hiệu quả. Các giải pháp hiện có tuy phổ biến nhưng việc tự xây dựng và làm chủ công nghệ truyền tải thời gian thực (real-time) và truyền phát video (streaming) vẫn là một thách thức kỹ thuật quan trọng nhằm tối ưu hóa trải nghiệm người dùng trong các điều kiện mạng khác nhau.

Để giải quyết bài toán này, đồ án lựa chọn phát triển hệ thống theo kiến trúc Client-Server. Hệ thống sử dụng các giao thức truyền tải thời gian thực như WebSocket và WebRTC để xử lý các tác vụ nhắn tin và gọi điện, đảm bảo sự tương tác tức thì giữa người dùng. Điểm nổi bật của đồ án là việc tích hợp công nghệ HLS (HTTP Live Streaming) vào quy trình xử lý nội dung video. Việc sử dụng HLS giúp chia nhỏ nội dung video thành các phân đoạn, giúp tối ưu hóa việc truyền tải và cho phép người dùng xem video mượt mà với độ trễ thấp, thích ứng tốt với băng thông mạng thay đổi.

Giải pháp được hiện thực hóa thông qua một ứng dụng web hoàn chỉnh với đầy đủ các chức năng: đăng ký, đăng nhập, tìm kiếm bạn bè, nhắn tin đa phương tiện và gọi điện video. Việc áp dụng HLS cho thấy hiệu quả rõ rệt trong việc cải thiện việc truyền tải và xem lại các video trong cuộc trò chuyện.

Kết quả đạt được là một hệ thống nhắn tin, gọi điện hoạt động ổn định, đáp ứng tốt các yêu cầu về tính năng thời gian thực và xử lý đa phương tiện. Đồ án đóng góp một mô hình tham khảo hiệu quả cho việc kết hợp giữa các dịch vụ real-time và kỹ thuật streaming hiện đại trong phát triển ứng dụng web.

Sinh viên thực hiện

(Ký và ghi rõ họ tên)

ABSTRACT

In the context of robust development in information technology, the demand for online communication via messaging and video calls is rapidly increasing. This necessitates applications that ensure high speed, low latency, and effective multimedia processing. While existing solutions are widespread, self-building and mastering real-time transmission and video streaming technologies remain significant technical challenges for optimizing user experience under varying network conditions.

To address this problem, this project implements a system based on Client-Server architecture. The system utilizes real-time protocols such as WebSocket and WebRTC to handle messaging and calling tasks, ensuring instant user interaction. A key highlight of the project is the integration of HLS (HTTP Live Streaming) technology for video content processing. HLS segments video content, optimizing transmission and allowing users to experience smooth playback with low latency, adapting effectively to network bandwidth fluctuations.

The solution is realized as a comprehensive web application featuring registration, authentication, friend search, multimedia messaging, and video calling. The application of HLS demonstrates clear effectiveness in improving video transmission and playback within conversations.

The result is a stable messaging and calling system that satisfies requirements for real-time performance and multimedia processing. The project provides an effective reference model for combining real-time services with modern streaming techniques in web application development.

MỤC LỤC

DANH MỤC HÌNH VẼ

DANH MỤC BẢNG BIỂU

DANH MỤC THUẬT NGỮ VÀ TỪ VIẾT TẮT

Thuật ngữ	**Ý nghĩa**
API	Giao diện lập trình ứng dụng (Application Programming Interface)
EUD	Phát triển ứng dụng người dùng cuối(End-User Development)
GWT	Công cụ lập trình Javascript bằng Java của Google (Google Web Toolkit)
HTML	Ngôn ngữ đánh dấu siêu văn bản (HyperText Markup Language)
IaaS	Dịch vụ hạ tầng

CHƯƠNG 1. GIỚI THIỆU ĐỀ TÀI

1.1 Đặt vấn đề

Trong kỷ nguyên số hóa hiện nay, nhu cầu kết nối và trao đổi thông tin trực tuyến đóng vai trò then chốt trong mọi hoạt động kinh tế - xã hội. Sự bùng nổ của các ứng dụng nhắn tin và gọi điện theo thời gian thực (Real-time Communication) đã thay đổi hoàn toàn thói quen giao tiếp của con người, xóa bỏ rào cản về khoảng cách địa lý.

Tuy nhiên, việc phụ thuộc vào các nền tảng thương mại có sẵn đôi khi đặt ra những lo ngại về quyền riêng tư dữ liệu, khả năng tùy biến cho các nhu cầu đặc thù của tổ chức, hoặc đơn giản là giới hạn trong việc tích hợp sâu vào các hệ thống nghiệp vụ nội bộ. Hơn nữa, dưới góc độ kỹ thuật, việc xây dựng một hệ thống truyền tải dữ liệu đa phương tiện (âm thanh, hình ảnh) với độ trễ thấp và tính ổn định cao luôn là một thách thức lớn, đòi hỏi sự am hiểu sâu sắc về các giao thức mạng hiện đại.

Xuất phát từ thực tế đó, đề tài tập trung nghiên cứu và xây dựng ứng dụng web nhắn tin, gọi điện nhằm giải quyết bài toán về làm chủ công nghệ truyền tin thời gian thực, đồng thời cung cấp một giải pháp giao tiếp trực tuyến hiệu quả, linh hoạt và có khả năng mở rộng cao.

1.2 Mục tiêu và phạm vi đề tài

Mục tiêu chính của đề tài là nghiên cứu và làm chủ các công nghệ cốt lõi trong việc xây dựng hệ thống giao tiếp thời gian thực, từ đó phát triển một ứng dụng web hoàn chỉnh hỗ trợ nhắn tin và gọi điện video. Cụ thể, đề tài hướng tới các mục tiêu sau:

- Nghiên cứu cơ chế hoạt động của giao thức WebSocket để thực hiện tính năng nhắn tin tức thời (Instant Messaging).
- Tìm hiểu và ứng dụng công nghệ WebRTC (Web Real-Time Communication) để triển khai tính năng gọi thoại và gọi video trực tiếp trên trình duyệt mà không cần cài đặt thêm phần mềm hỗ trợ.
- Xây dựng hệ thống backend có khả năng xử lý đồng thời nhiều kết nối, đảm bảo tính ổn định và tốc độ truyền tải nhanh.

Về phạm vi, đề tài tập trung vào việc phát triển các chức năng thiết yếu nhất của một ứng dụng giao tiếp hiện đại:

- **Quản lý tài khoản:** Đăng ký, đăng nhập, xác thực người dùng an toàn.
- **Chức năng nhắn tin:** Gửi nhận tin nhắn văn bản, hình ảnh và tệp tin theo thời gian thực; cập nhật trạng thái tin nhắn.
- **Chức năng gọi điện:** Thiết lập cuộc gọi thoại và gọi video cá nhân (1-1) với chất lượng ổn định.
- **Quản lý danh bạ:** Tìm kiếm người dùng, kết bạn và tạo nhóm trò chuyện cơ bản.

Sản phẩm được triển khai trên nền tảng Web, hướng tới sự tương thích tốt với các trình duyệt phổ biến hiện nay như Google Chrome, Microsoft Edge và Firefox.

1.3 Định hướng giải pháp

Để giải quyết các vấn đề đã nêu và đạt được mục tiêu đề ra, đồ án lựa chọn hướng tiếp cận phát triển ứng dụng Web dựa trên kiến trúc Client-Server kết hợp với mô hình Peer-to-Peer cho tính năng gọi điện. Cụ thể:

Về mặt công nghệ, hệ thống sẽ được xây dựng dựa trên nền tảng Node.js, một môi trường chạy JavaScript mạnh mẽ cho phía server, cho phép xử lý lượng lớn kết nối đồng thời với độ trễ thấp. Giao thức WebSocket (thông qua thư viện Socket.io) sẽ được sử dụng làm nòng cốt cho việc truyền tải tin nhắn tức thời và các tín hiệu điều khiển cuộc gọi (signaling).

Đối với tính năng gọi thoại và video, đồ án sử dụng công nghệ WebRTC. Đây là công nghệ nguồn mở cho phép các trình duyệt giao tiếp trực tiếp với nhau (Peer-to-Peer) để truyền tải luồng dữ liệu âm thanh và hình ảnh mà không cần thông qua server trung gian để chuyển tiếp dữ liệu media, giúp giảm tải cho server và tăng tốc độ truyền tin.

Phía người dùng (Frontend) sẽ được phát triển bằng thư viện ReactJS để đảm bảo giao diện trực quan, tương tác mượt mà và trải nghiệm người dùng tốt nhất. Cơ sở dữ liệu MongoDB sẽ được sử dụng để lưu trữ thông tin người dùng và lịch sử tin nhắn nhờ tính linh hoạt và khả năng mở rộng cao.

Đóng góp chính của đồ án là việc tích hợp thành công các công nghệ trên vào một sản phẩm thực tế, chứng minh khả năng làm chủ kỹ thuật lập trình mạng phức tạp và cung cấp một giải pháp thay thế khả thi cho các nhu cầu giao tiếp nội bộ hoặc quy mô nhỏ.

1.4 Bố cục đồ án

Phần còn lại của báo cáo đồ án tốt nghiệp được tổ chức như sau. Chương 2 tập trung khảo sát các hệ thống tương tự hiện có và phân tích chi tiết các yêu cầu chức năng, phi chức năng của hệ thống cần xây dựng. Chương 3 trình bày cơ sở lý thuyết về các công nghệ chủ chốt được sử dụng trong đề tài, bao gồm Node.js, WebSocket và giao thức WebRTC. Tiếp theo, Chương 4 đi sâu vào thiết kế kiến trúc hệ thống, thiết kế cơ sở dữ liệu, giao diện người dùng, đồng thời mô tả quá trình triển khai và đánh giá kết quả thực nghiệm. Chương 5 thảo luận về các giải pháp kỹ thuật cụ thể và những đóng góp nổi bật của đề tài trong việc tối ưu hóa hiệu năng hoặc trải nghiệm người dùng. Cuối cùng, Chương 6 tổng kết lại những kết quả đã đạt được, nhìn nhận các hạn chế còn tồn tại và đề xuất hướng phát triển tiếp theo cho hệ thống.

CHƯƠNG 2. KHẢO SÁT VÀ PHÂN TÍCH YÊU CẦU

Chương này sẽ tập trung vào việc khảo sát các hệ thống tương tự, phân tích các yêu cầu chức năng và phi chức năng của hệ thống Chat App. Từ việc khảo sát, đồ án sẽ đi sâu vào thiết kế tổng quan các chức năng thông qua biểu đồ Use Case, đồng thời đặc tả chi tiết các quy trình nghiệp vụ chính để làm cơ sở cho việc thiết kế và cài đặt ở các chương sau.

2.1 Khảo sát hiện trạng

Hiện nay, nhu cầu trao đổi thông tin trực tuyến đóng vai trò thiết yếu trong đời sống và công việc. Các ứng dụng nhắn tin tức thời (Instant Messaging) không chỉ đơn thuần là gửi văn bản mà còn tích hợp đa phương tiện, gọi thoại/video chất lượng cao và khả năng làm việc nhóm.

Qua khảo sát các ứng dụng phổ biến trên thị trường như Facebook Messenger, Zalo, và Telegram, nhóm nhận thấy các ứng dụng này đều có điểm chung là tập trung vào trải nghiệm người dùng mượt mà, tốc độ phản hồi nhanh (real-time) và khả năng xử lý dữ liệu đa phương tiện tốt. Tuy nhiên, việc xây dựng một hệ thống tương tự đòi hỏi sự kết hợp phức tạp của nhiều công nghệ hiện đại.

Dựa trên khảo sát và mục tiêu nghiên cứu, đồ án tập trung xây dựng một ứng dụng Chat trên nền web với các nhóm tính năng quan trọng sau:

- **Xác thực & Người dùng**: Đảm bảo bảo mật tài khoản, hỗ trợ đăng nhập đa nền tảng (OAuth2 Google), quản lý thông tin cá nhân và trạng thái hoạt động (Online/Offline).
- **Mạng xã hội (Social)**: Xây dựng mạng lưới kết nối thông qua tìm kiếm, kết bạn, quản lý danh sách bạn bè và tạo nhóm chat.
- **Nhắn tin (Messaging)**: Hỗ trợ trao đổi tin nhắn văn bản, biểu cảm (Emoji), và đặc biệt là chia sẻ tài liệu, hình ảnh, video (hỗ trợ Streaming HLS cho video dung lượng lớn).
- **Gọi điện (Calling)**: Tích hợp tính năng gọi Video Call trực tiếp giữa người dùng thông qua công nghệ WebRTC.
- **Hệ thống**: Các tính năng nền tảng như thông báo thời gian thực (Notification) và tìm kiếm toàn cục.

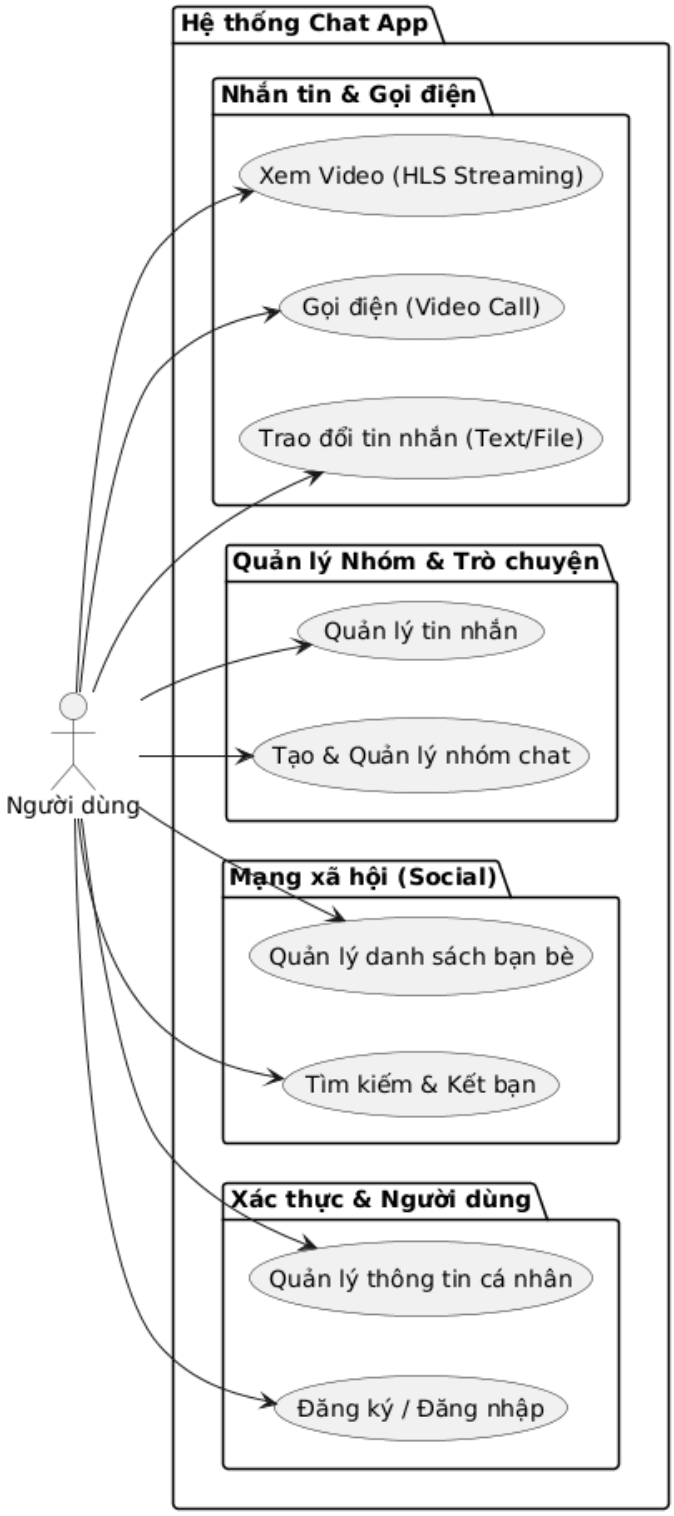
2.2 Tổng quan chức năng

Phần này tóm tắt các nhóm chức năng chính mà hệ thống hướng tới, đáp ứng nhu cầu giao tiếp toàn diện của người dùng. Hệ thống không chỉ cung cấp các tính năng nhắn tin cơ bản mà còn tích hợp các tiện ích nâng cao như gọi video chất lượng cao, chia sẻ tệp tin dung lượng lớn và quản lý nhóm hiệu quả. Các chức năng được chia thành các phân hệ rõ

ràng để thuận tiện cho việc thiết kế và phát triển.

2.2.1 Biểu đồ use case tổng quát

Hệ thống được thiết kế xoay quanh tác nhân chính là **Người dùng (User)**. Dưới đây là biểu đồ Use Case tổng quát mô tả các chức năng chính của hệ thống:



Hình 2.1: Biểu đồ Use Case tổng quát của hệ thống

Danh sách các tác nhân:

- **Người dùng (User)**: Là người đã đăng ký tài khoản thành công và đăng nhập vào hệ thống. Người dùng có toàn quyền sử dụng các chức năng giao tiếp và quản lý cá nhân.

Mô tả các nhóm Use Case chính:

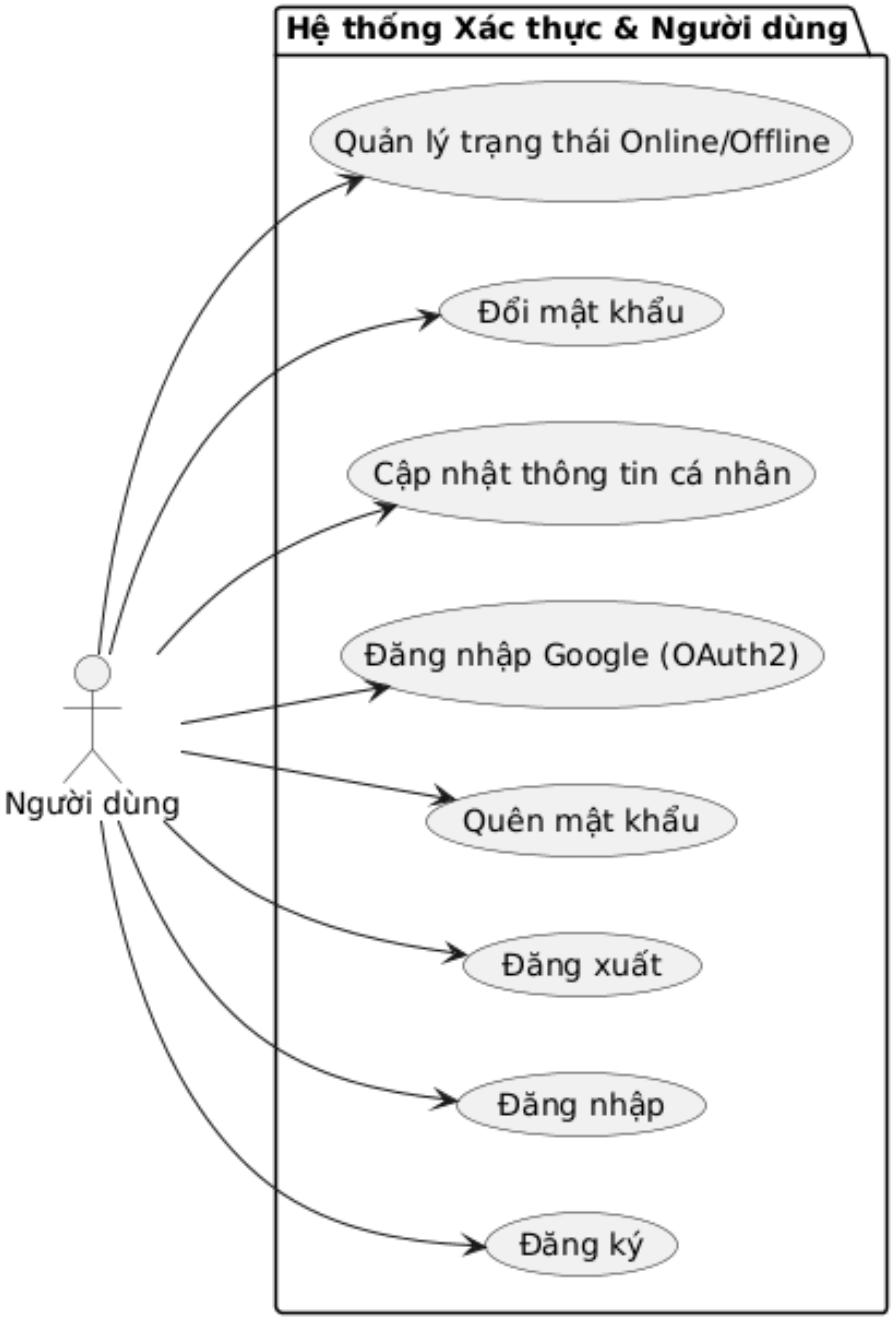
1. **Nhóm Xác thực (Authentication)**:
    - Đăng ký, Đăng nhập (tài khoản thường và Google OAuth2).
    - Quên mật khẩu, Đổi mật khẩu.
    - Cập nhật Profile (Avatar, thông tin cá nhân) và quản lý trạng thái Online/Offline.
2. **Nhóm Bạn bè & Tìm kiếm (Social)**:
    - Tìm kiếm người dùng khác trong hệ thống.
    - Gửi lời mời kết bạn, Chấp nhận hoặc Từ chối lời mời.
    - Xem danh sách bạn bè và Hủy kết bạn.
3. **Nhóm Nhóm & Trò chuyện (Group & Chat)**:
    - Tạo cuộc hội thoại mới hoặc mở cuộc hội thoại đã có.
    - Tạo nhóm chat mới, xem danh sách nhóm.
    - Quản lý thành viên nhóm (Thêm, Xóa thành viên).
    - Xóa cuộc hội thoại lịch sử.
4. **Nhóm Nhắn tin & Gọi điện (Messaging & Calling)**:
    - Gửi nhận tin nhắn văn bản (Text), biểu tượng cảm xúc (Emoji).
    - Gửi nhận tệp tin đa phương tiện: Ảnh, Video, Audio.
    - Xem video chất lượng cao thông qua luồng phát trực tuyến (HLS Streaming).
    - Thực hiện cuộc gọi Video trực tuyến (Video Call) sử dụng WebRTC.

2.2.2 Biểu đồ use case phân rã

Để làm rõ hơn các chức năng của hệ thống, phần này sẽ trình bày chi tiết các biểu đồ use case phân rã cho từng phân hệ chính.

a, Phân hệ Xác thực & Người dùng

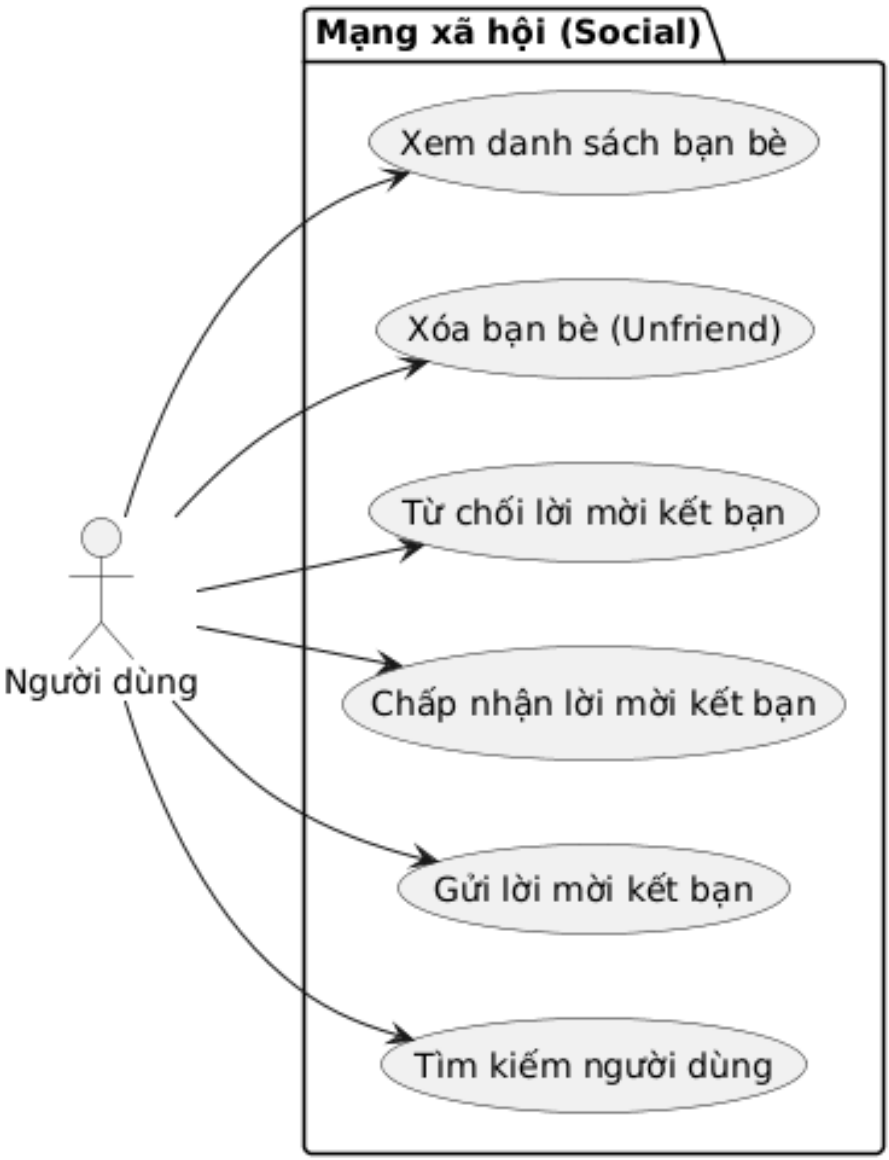
Phân hệ này quản lý việc truy cập và thông tin cá nhân của người dùng. Các chức năng bao gồm Đăng ký, Đăng nhập (hỗ trợ cả tài khoản thường và Google OAuth2), Đăng xuất, Quên mật khẩu. Ngoài ra, người dùng có thể cập nhật thông tin cá nhân (Avatar, thông tin cơ bản) và quản lý trạng thái online/offline của mình.



Hình 2.2: Biểu đồ Use Case phân rã: Xác thực & Người dùng

b, Phân hệ Mạng xã hội (Social)

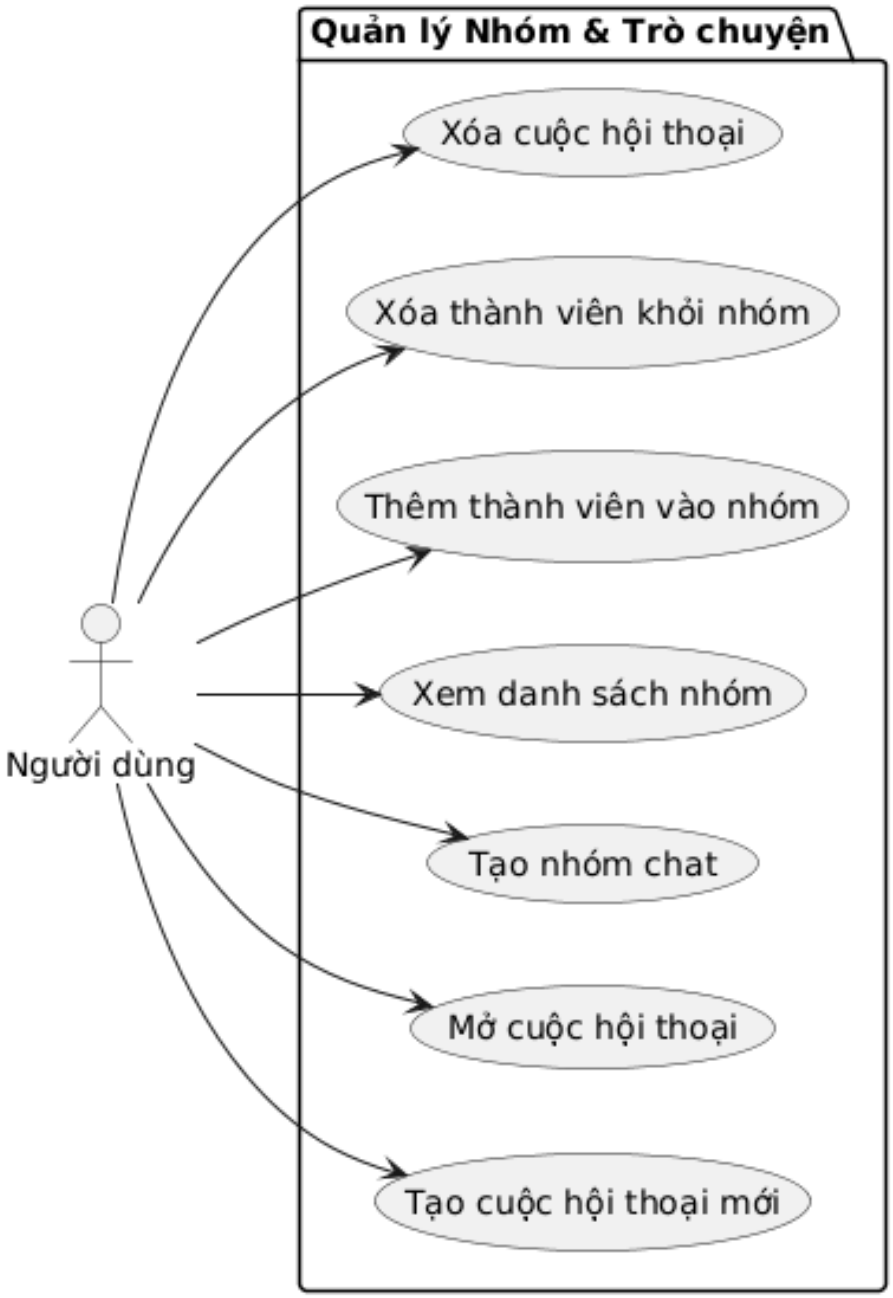
Phân hệ này tập trung vào tính năng kết nối giữa các người dùng. Người dùng có thể tìm kiếm người khác trong hệ thống, gửi lời mời kết bạn, và quản lý danh sách bạn bè (chấp nhận, từ chối hoặc hủy kết bạn).



Hình 2.3: Biểu đồ Use Case phân rã: Mạng xã hội

c, Phân hệ Quản lý Nhóm & Trò chuyện

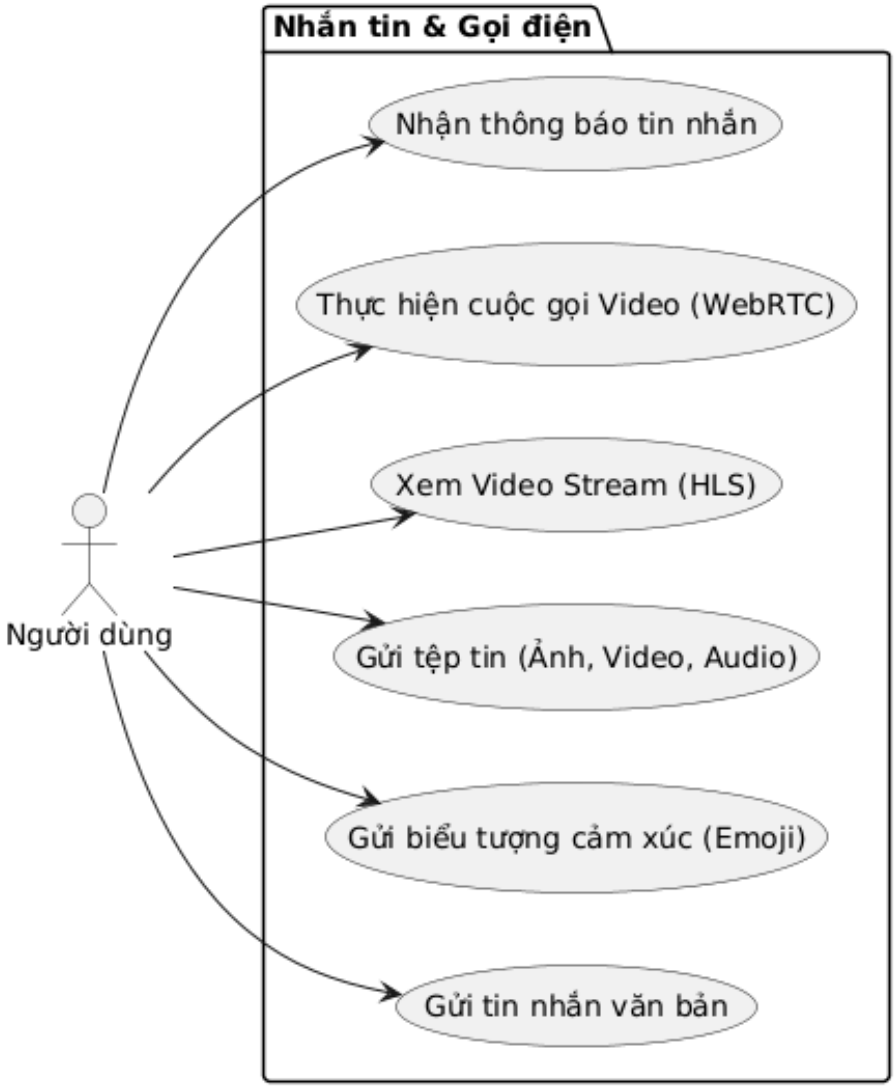
Phân hệ này cho phép người dùng quản lý các cuộc hội thoại và nhóm chat. Người dùng có thể tạo cuộc hội thoại mới, xem danh sách nhóm, tạo nhóm mới, và quản lý thành viên trong nhóm (thêm hoặc xóa thành viên).



Hình 2.4: Biểu đồ Use Case phân rã: Quản lý Nhóm & Trò chuyện

d, Phân hệ Nhắn tin & Gọi điện

Đây là phân hệ cốt lõi của ứng dụng, cung cấp khả năng trao đổi thông tin đa phương tiện. Người dùng có thể gửi tin nhắn văn bản, emoji, và các tệp tin (ảnh, video, audio). Hệ thống hỗ trợ xem video chất lượng cao qua HLS streaming và thực hiện cuộc gọi video thời gian thực qua WebRTC.



Hình 2.5: Biểu đồ Use Case phân rã: Nhắn tin & Gọi điện

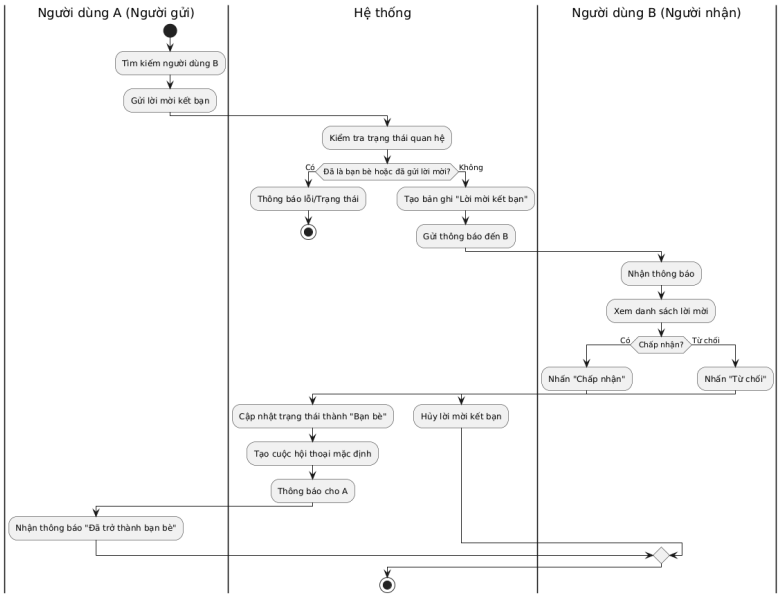
2.2.3 Quy trình nghiệp vụ

Nếu sản phẩm/hệ thống cần xây dựng có quy trình nghiệp vụ quan trọng/đáng chú ý, sinh viên cần mô tả và vẽ biểu đồ hoạt động minh họa quy trình nghiệp vụ đó. Sinh viên lưu ý đây không phải là luồng sự kiện của từng use case, mà là luồng hoạt động kết hợp nhiều use case để thực hiện một nghiệp vụ nào đó.

Đồ án tập trung vào hai quy trình nghiệp vụ tiêu biểu đại diện cho logic nghiệp vụ (Kết bạn) và logic kỹ thuật phức tạp (Gọi Video).

a, Quy trình Kết bạn (Friend Request Process)

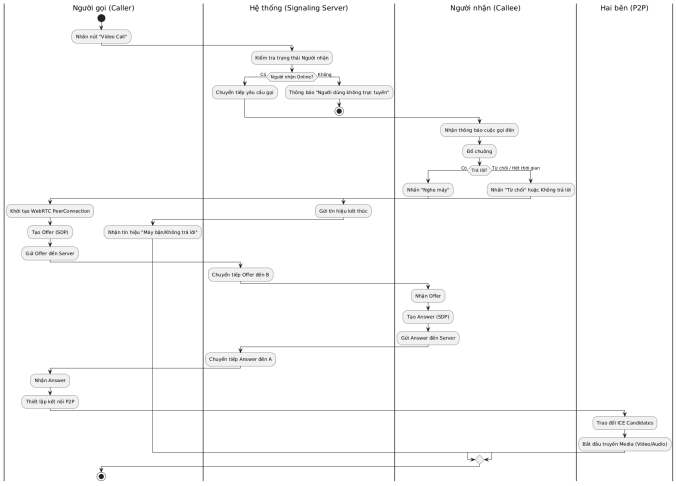
Đây là quy trình cơ bản để mở rộng mạng lưới xã hội của người dùng. Quy trình bắt đầu từ việc người dùng tìm kiếm, gửi lời mời, và kết thúc khi hai người trở thành bạn bè và có thể nhắn tin cho nhau.



Hình 2.6: Biểu đồ hoạt động: Quy trình Kết bạn

b, Quy trình thực hiện cuộc gọi Video (Video Call Process)

Quy trình này minh họa sự tương tác phức tạp giữa hai nười dùng và hệ thống (Signaling Server) để thiết lập một phiên kết nối thời gian thực thông qua WebRTC.



Hình 2.7: Biểu đồ hoạt động: Quy trình gọi Video Call

2.3 Đặc tả chức năng

Dưới đây là đặc tả chi tiết cho 5 use case quan trọng nhất của hệ thống, bao gồm: Đăng nhập, Tìm kiếm người dùng, Tạo nhóm chat, Gửi tin nhắn và Gọi Video.

2.3.1 Đặc tả Use Case: Đăng nhập (Login)

Tên Use Case	**Đăng nhập hệ thống**
Mô tả	Cho phép người dùng truy cập vào hệ thống để sử dụng các tính năng.
Tiền điều kiện	Người dùng đã có tài khoản (đã đăng ký).
Luồng sự kiện chính	1. Người dùng truy cập trang đăng nhập. 2. Người dùng nhập Email và Mật khẩu. 3. Người dùng nhấn nút "Đăng nhập". 4. Hệ thống kiểm tra thông tin xác thực. 5. Nếu đúng, hệ thống chuyển hướng vào trang chủ Chat.
Luồng ngoại lệ	4a. Nếu thông tin sai: Hệ thống báo lỗi "Email hoặc mật khẩu không đúng". 4b. Nếu tài khoản bị khóa: Thông báo liên hệ quản trị viên.
Hậu điều kiện	Người dùng truy cập thành công, trạng thái chuyển sang Online.

Bảng 2.1: Đặc tả Use Case Đăng nhập

2.3.2 Đặc tả Use Case: Tìm kiếm người dùng (Search User)

Tên Use Case	**Tìm kiếm người dùng**
Mô tả	Người dùng tìm kiếm người khác trong hệ thống thông qua tên hoặc email để kết bạn.
Tiền điều kiện	Người dùng đã đăng nhập.
Luồng sự kiện chính	1. Người dùng chọn chức năng tìm kiếm. 2. Người dùng nhập từ khóa (Tên hoặc Email). 3. Hệ thống hiển thị danh sách kết quả phù hợp. 4. Người dùng chọn người cần kết nối để xem thông tin hoặc nhắn tin.
Hậu điều kiện	Danh sách kết quả tìm kiếm được hiển thị.

Bảng 2.2: Đặc tả Use Case Tìm kiếm người dùng

2.3.3 Đặc tả Use Case: Tạo nhóm chat (Create Group)

Tên Use Case	**Tạo nhóm chat**
Mô tả	Tạo một cuộc trò chuyện nhóm mới với nhiều thành viên.
Tiền điều kiện	Người dùng đã đăng nhập và là bạn bè với các thành viên muốn thêm.
Luồng sự kiện chính	1. Người dùng nhấn nút "Tạo nhóm". 2. Hệ thống hiển thị danh sách bạn bè. 3. Người dùng chọn các thành viên và đặt tên nhóm. 4. Người dùng nhấn "Tạo". 5. Hệ thống tạo nhóm và thêm các thành viên vào.
Hậu điều kiện	Nhóm chat mới được tạo, các thành viên nhận được thông báo.

Bảng 2.3: Đặc tả Use Case Tạo nhóm chat

2.3.4 Đặc tả Use Case: Gửi tin nhắn (Send Message)

Tên Use Case	**Gửi tin nhắn (Văn bản/Đa phương tiện)**
Mô tả	Gửi nội dung tin nhắn đến một người dùng hoặc một nhóm.
Tiền điều kiện	Người dùng đã vào giao diện chat với đối phương.
Luồng sự kiện chính	1. Người dùng nhập nội dung hoặc chọn tệp tin (Ảnh/Video). 2. Người dùng nhấn Gửi (Enter). 3. Hệ thống nhận tin nhắn, lưu vào CSDL. 4. Hệ thống gửi tin nhắn đến người nhận qua WebSocket. 5. Giao diện người dùng cập nhật tin nhắn mới vừa gửi.
Hậu điều kiện	Tin nhắn hiển thị ở cả hai phía người gửi và người nhận.

Bảng 2.4: Đặc tả Use Case Gửi tin nhắn

2.3.5 Đặc tả Use Case: Gọi Video (Video Call)

Tên Use Case	**Thực hiện cuộc gọi Video Call**
Mô tả	Thiết lập cuộc gọi video trực tiếp giữa hai người dùng (P2P).
Tiền điều kiện	Cả hai người dùng đều đang Online. Trình duyệt hỗ trợ WebRTC.
Luồng sự kiện chính	1. Người dùng A nhấn nút Video Call. 2. Hệ thống gửi thông báo cuộc gọi đến Người dùng B. 3. Người dùng B chấp nhận cuộc gọi. 4. Hệ thống thiết lập kết nối Peer-to-Peer (WebRTC) giữa A và B. 5. Hai bên trao đổi luồng Media (Video/Audio). 6. Một trong hai bên nhấn nút Kết thúc để dừng cuộc gọi.
Luồng ngoại lệ	2a. Người dùng B Offline: Hệ thống báo không liên lạc được. 3a. Người dùng B từ chối: Hệ thống báo máy bận.
Hậu điều kiện	Cuộc gọi kết thúc, lịch sử cuộc gọi được lưu lại.

Bảng 2.5: Đặc tả Use Case Gọi Video

2.4 Yêu cầu phi chức năng

Các yêu cầu phi chức năng đóng vai trò quan trọng trong việc đảm bảo chất lượng và trải nghiệm người dùng của hệ thống. Dưới đây là các yêu cầu chi tiết:

- **Hiệu năng (Performance)**:
  - Hệ thống phải đảm bảo tính thời gian thực (Real-time). Độ trễ khi gửi/nhận tin nhắn không được vượt quá 200ms trong điều kiện mạng ổn định.
  - Video streaming phải mượt mà, hỗ trợ tự động điều chỉnh chất lượng (Adaptive Bitrate Streaming) tùy theo băng thông mạng của người dùng.
  - Thời gian tải trang ban đầu không quá 3 giây.
- **Bảo mật (Security)**:

  - Tài khoản người dùng phải được bảo vệ, mật khẩu lưu trữ trong cơ sở dữ liệu phải được mã hóa (hashing).
  - Giao tiếp giữa Client và Server phải được mã hóa thông qua giao thức HTTPS và WSS (Secure WebSocket).
  - Cơ chế xác thực qua Token (JWT) phải đảm bảo tính toàn vẹn và hết hạn hợp lý.
  - Dữ liệu riêng tư của người dùng không được rò rỉ hoặc truy cập trái phép.

- **Tính khả dụng và Trải nghiệm người dùng (Usability)**:
  - Giao diện phải thân thiện, dễ sử dụng, hỗ trợ Responsive để hiển thị tốt trên cả máy tính và thiết bị di động.
  - Hệ thống phải có cơ chế phản hồi rõ ràng cho các hành động của người dùng (ví dụ: thông báo loading, thông báo lỗi, thông báo thành công).

- **Độ tin cậy (Reliability)**:
  - Hệ thống hoạt động ổn định 24/7.
  - Dữ liệu tin nhắn và tệp tin không được thất lạc trong quá trình truyền tải.
  - Hệ thống có khả năng phục hồi sau khi gặp sự cố gián đoạn kết nối mạng.

- **Khả năng bảo trì và mở rộng (Maintainability & Scalability)**:
  - Mã nguồn phải được tổ chức rõ ràng theo kiến trúc định sẵn (ví dụ: Client-Server, MVC) để dễ dàng nâng cấp và sửa lỗi.
  - Hệ thống có khả năng mở rộng để hỗ trợ thêm tính năng mới mà không làm ảnh hưởng đến các chức năng hiện có.
  - Sử dụng các công nghệ containerization (Docker) để dễ dàng triển khai trên các môi trường khác nhau.

CHƯƠNG 3. CÔNG NGHỆ SỬ DỤNG

Chương này trình bày chi tiết về các công nghệ, nền tảng và công cụ được sử dụng để xây dựng và triển khai hệ thống. Việc lựa chọn công nghệ dựa trên các tiêu chí về hiệu năng thời gian thực, trải nghiệm người dùng, khả năng mở rộng (scalability) và tính ổn định của hệ thống.

3.1 Nền tảng Backend và Xử lý nghiệp vụ

3.1.1 Node.js và môi trường Runtime

Hệ thống sử dụng Node.js làm nền tảng runtime chính cho phía Server. Node.js được xây dựng trên V8 JavaScript engine của Chrome, sử dụng mô hình non-blocking I/O và event-driven [1]. Điều này đặc biệt phù hợp cho các ứng dụng thời gian thực cần xử lý hàng nghìn kết nối đồng thời với độ trễ thấp như chat và gọi điện. Kết hợp với framework Express.js, Node.js cung cấp khả năng xây dựng các API RESTful hiệu quả và dễ dàng mở rộng.

3.1.2 Prisma ORM

Để tương tác với cơ sở dữ liệu MongoDB một cách an toàn và tối ưu, đồ án sử dụng Prisma ORM. Prisma cung cấp một layer trừu tượng hóa giúp định nghĩa schema dữ liệu rõ ràng, hỗ trợ truy vấn type-safe (an toàn kiểu dữ liệu) và tự động tạo ra các migration [2]. Việc này giúp giảm thiểu lỗi runtime và tăng tốc độ phát triển so với việc sử dụng driver MongoDB thuần túy hoặc Mongoose.

3.1.3 Xử lý dữ liệu và Validation với Zod

Để đảm bảo tính toàn vẹn của dữ liệu đầu vào từ phía Client, hệ thống tích hợp thư viện Zod cho việc validation (kiểm thực). Zod cho phép định nghĩa các schema xác thực chặt chẽ cho từng API endpoint, tự động loại bỏ các trường dữ liệu thừa và báo lỗi chi tiết khi dữ liệu không đúng định dạng.

3.1.4 Xử lý Đa phương tiện với FFmpeg

Thư viện FFmpeg được tích hợp vào Backend để thực hiện các tác vụ xử lý video và hình ảnh phức tạp. Cụ thể, FFmpeg đóng vai trò cốt lõi trong việc chuyển đổi định dạng video (transcoding) sang chuẩn HLS, tạo hình ảnh thu nhỏ (thumbnail) và tối ưu hóa kích thước file trước khi lưu trữ, đảm bảo trải nghiệm xem video mượt mà trên nhiều thiết bị.

3.2 Nền tảng Frontend và Giao diện người dùng

3.2.1 ReactJS

Giao diện người dùng (Client) được xây dựng dựa trên thư viện ReactJS (phiên bản 19). Với kiến trúc dựa trên Component và Virtual DOM, React cho phép xây dựng ứng dụng Single Page Application (SPA) với trải nghiệm người dùng mượt mà, phản hồi nhanh và dễ dàng tái sử dụng mã nguồn cho các chức năng tương tự nhau.

3.2.2 Vite

Thay vì sử dụng các công cụ build truyền thống như Webpack, dự án sử dụng Vite - một build tool thế hệ mới với tốc độ vượt trội. Vite tận dụng khả năng của ES Modules trong trình duyệt hiện đại để hỗ trợ tính năng Hot Module Replacement (HMR) tức thì, giúp giảm đáng kể thời gian chờ đợi trong quá trình phát triển và tối ưu hóa bundle size khi build production.

3.2.3 Quản lý trạng thái với Zustand

Để quản lý state (trạng thái) phức tạp của ứng dụng chat (ví dụ: danh sách tin nhắn, trạng thái online/offline, thông báo), hệ thống sử dụng thư viện Zustand. Đây là giải pháp quản lý state nhỏ gọn, hiệu năng cao và đơn giản hơn so với Redux, giúp chia sẻ dữ liệu giữa các component một cách dễ dàng và tránh việc render lại không cần thiết (re-render).

3.2.4 TailwindCSS và Shadcn UI

Giao diện được thiết kế hiện đại và nhất quán nhờ sự kết hợp giữa TailwindCSS và Shadcn UI. TailwindCSS cung cấp bộ công cụ "utility-first" giúp styling nhanh chóng, trong khi Shadcn UI cung cấp các component giao diện cao cấp (như Modal, Dialog, Toast) được xây dựng trên nền tảng Radix UI, đảm bảo tính thẩm mỹ và khả năng truy cập (accessibility).

3.3 Giao thức truyền tải và Thời gian thực

3.3.1 WebSocket và Socket.io

Giao thức WebSocket được sử dụng để duy trì kết nối hai chiều liên tục giữa Client và Server. Thư viện Socket.io được lựa chọn để triển khai WebSocket nhờ tính ổn định cao, khả năng tự động fallback về HTTP long-polling khi mạng kém và hỗ trợ quản lý Room/Namespace hiệu quả cho tính năng chat nhóm.

3.3.2 WebRTC (Web Real-Time Communication)

Tính năng gọi thoại và gọi video P2P (Peer-to-Peer) được xây dựng trên công nghệ WebRTC [3]. WebRTC cho phép truyền tải trực tiếp âm thanh và hình ảnh giữa các trình duyệt mà không cần qua server trung gian (trừ quá trình signaling), đảm bảo độ trễ thấp nhất và bảo mật cao thông qua mã hóa DTLS/SRTP.

3.3.3 Giao thức HLS (HTTP Live Streaming)

Để giải quyết bài toán truyền tải video chất lượng cao, hệ thống áp dụng giao thức HLS. Video tải lên được chia nhỏ thành các phân đoạn (segment) ‘.ts‘ ngắn và được quản lý bởi một file manifest ‘.m3u8‘. Công nghệ này cho phép trình duyệt tải từng phần video theo nhu cầu (buffering), hỗ trợ Adaptive Bitrate Streaming (ABR) để tự động điều chỉnh chất lượng video dựa trên băng thông mạng của người dùng.

3.4 Lưu trữ và Cơ sở dữ liệu

3.4.1 MongoDB

MongoDB đóng vai trò là Primary Database, lưu trữ toàn bộ thông tin người dùng, tin nhắn và quan hệ bạn bè. Cấu trúc Document linh hoạt (JSON-like) của MongoDB rất phù hợp với dữ liệu chat đa dạng và có cấu trúc thay đổi thường xuyên [4].

3.4.2 Redis

Redis được sử dụng như một lớp Cache và Message Broker hiệu năng cao. Hệ thống dùng Redis để lưu trữ session đăng nhập, danh sách người dùng online và làm Adapter cho Socket.io để hỗ trợ scale hệ thống sang nhiều instance, đồng thời giảm tải truy vấn trực tiếp vào MongoDB.

3.4.3 AWS S3 (Simple Storage Service)

Toàn bộ dữ liệu đa phương tiện (ảnh, video, tệp tin) được lưu trữ trên dịch vụ đám mây AWS S3 thay vì ổ cứng local. AWS S3 đảm bảo độ bền dữ liệu, khả năng truy xuất nhanh qua CDN và khả năng mở rộng dung lượng lữu trữ không giới hạn [5].

3.5 Hạ tầng và Triển khai (Infrastructure)

3.5.1 Docker và Docker Compose

Ứng dụng được đóng gói (Containerization) bằng Docker, bao gồm các container riêng biệt cho Backend, Frontend, Database và Redis. Docker Compose được sử dụng để định nghĩa và khởi chạy toàn bộ stack dịch vụ này một cách đồng bộ, đảm bảo tính nhất quán giữa môi trường Dev và Production [6].

3.5.2 Amazon EC2 và Nginx

Hệ thống được deploy trên máy chủ ảo Amazon EC2 chạy Ubuntu. Nginx đóng vai trò là Reverse Proxy và Web Server, chịu trách nhiệm xử lý SSL (HTTPS), phục vụ các file tĩnh của Frontend và điều hướng traffic API/Socket vào đúng port của container Backend [7].

3.5.3 Amazon SES (Simple Email Service)

Hệ thống tích hợp Amazon SES thông qua AWS SDK để thực hiện các tác vụ gửi email giao dịch (Transactional Email) như xác thực tài khoản (OTP) và khôi phục mật khẩu, đảm bảo tỷ lệ gửi tin thành công cao và không bị chặn bởi các bộ lọc spam [8].

CHƯƠNG 4. THIẾT KẾ, TRIỂN KHAI VÀ ĐÁNH GIÁ HỆ THỐNG

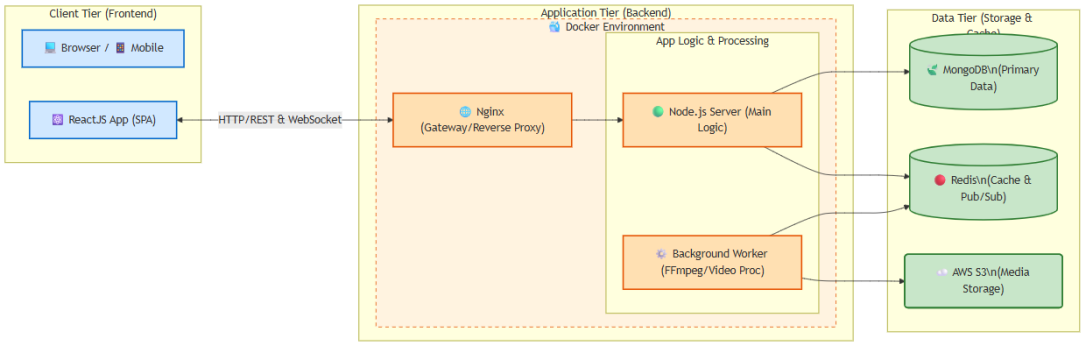
4.1 Thiết kế kiến trúc hệ thống

4.1.1 Mô hình tổng quan

Dựa trên yêu cầu về tính thời gian thực (real-time), khả năng xử lý phương tiện (media processing) và độ tin cậy, hệ thống được thiết kế theo **Kiến trúc 3 Tầng (3-Tier Architecture)** kết hợp với mô hình **Microservices** ở mức độ Containerization. Kiến trúc này tách biệt rõ ràng giữa giao diện người dùng, logic nghiệp vụ và lưu trữ dữ liệu.

Hệ thống bao gồm các thành phần chính tương tác như sau:

- **Client Layer (SPA):** Ứng dụng ReactJS chạy trên trình duyệt, kết nối đến Server qua REST API (cho các tác vụ stateless) và WebSocket (cho các tác vụ stateful/real-time).
- **Application Server Layer:** Node.js Server xử lý logic chính, bao gồm xác thực, điều phối tin nhắn và xử lý tín hiệu gọi điện (Signaling).
- **Background Worker Layer:** Các tiến trình nền (có thể chạy cùng Node.js hoặc tách riêng) phụ trách việc xử lý video nặng bằng FFmpeg để không làm chặn (blocking) luồng chính.
- **Data Layer:** Bao gồm MongoDB (Primary DB), Redis (Cache/PubSub) và AWS S3 (Object Storage).



Hình 4.1: Mô hình kiến trúc tổng quan 3 tầng (3-Tier Architecture)

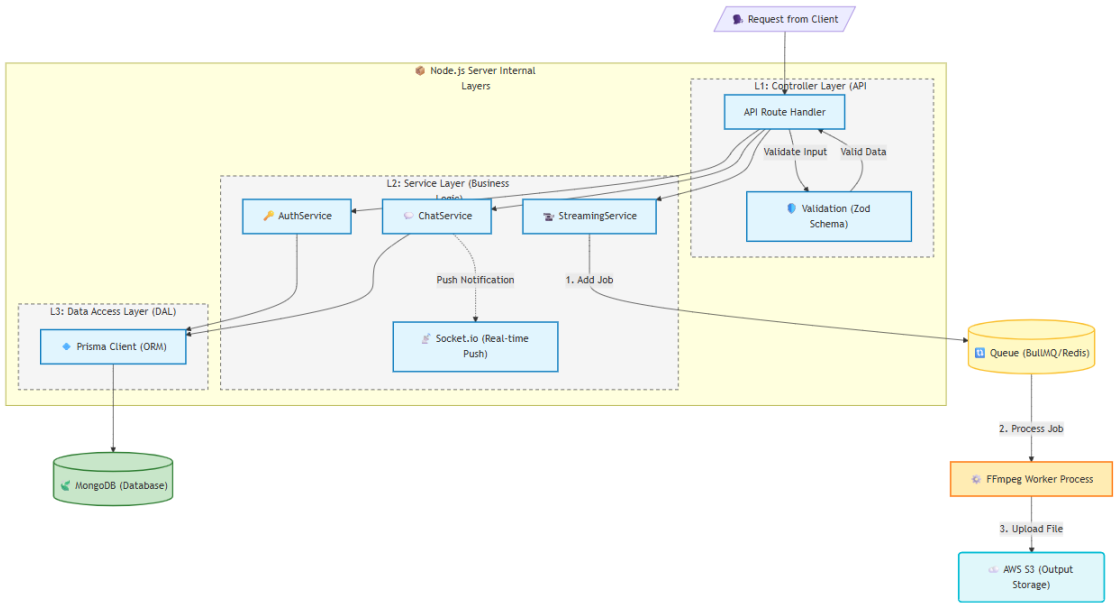
4.1.2 Kiến trúc chi tiết phía Backend

Backend được tổ chức theo mô hình **Layered Architecture** chặt chẽ:

1. **Router/Controller:** Tiếp nhận request, validate dữ liệu đầu vào bằng Zod, sau đó chuyển xuống Service.
2. **Service Layer:** Chứa toàn bộ business rules. Tại đây, các quy trình phức tạp như "Kiểm tra quyền bạn bè -> Tạo log tin nhắn -> Đẩy thông báo Socket" được thực thi.

3. **Data Access Layer (Repository):** Sử dụng Prisma ORM để tương tác với DB. Tầng này giúp ẩn giấu các câu lệnh query phức tạp.

Đặc biệt, module **Streaming Service** hoạt động độc lập: video sau khi upload sẽ được đưa vào hàng đợi (Queue), sau đó được transcode sang chuẩn HLS (m3u8) và upload lên S3.



Hình 4.2: Kiến trúc chi tiết Backend theo mô hình Layered

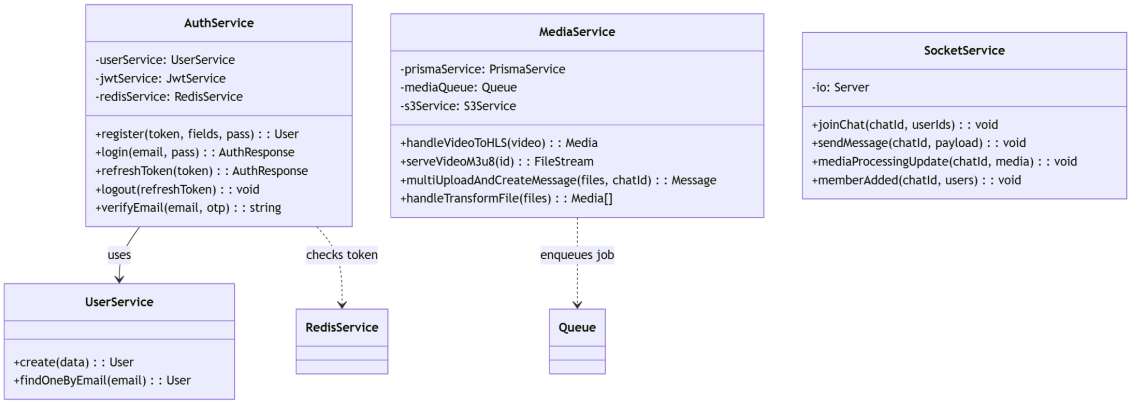
4.2 Thiết kế chi tiết

4.2.1 Thiết kế giao diện

Phần này có độ dài từ hai đến ba trang. Sinh viên đặc tả thông tin về màn hình mà ứng dụng của mình hướng tới, bao gồm độ phân giải màn hình, kích thước màn hình, số lượng màu sắc hỗ trợ, v.v. Tiếp đến, sinh viên đưa ra các thống nhất/chuẩn hóa của mình khi thiết kế giao diện như thiết kế nút, điều khiển, vị trí hiển thị thông điệp phản hồi, phối màu, v.v. Sau cùng sinh viên đưa ra một số hình ảnh minh họa thiết kế giao diện cho các chức năng quan trọng nhất. Lưu ý, sinh viên không nhầm lẫn giao diện thiết kế với giao diện của sản phẩm sau cùng.

4.2.2 Thiết kế lớp (Class Design)

Dựa trên kiến trúc Layered Architecture và Microservices, hệ thống được chia thành nhiều lớp xử lý chuyên biệt. Dưới đây là thiết kế chi tiết các lớp Service quan trọng nhất, nơi chứa phần lớn logic nghiệp vụ (Business Logic) của hệ thống.



Hình 4.3: Thiết kế lớp (Class Design)

a, Chi tiết các lớp xử lý chính

1. AuthService (Dịch vụ xác thực) Lớp **AuthService** chịu trách nhiệm quản lý phiên làm việc của người dùng, bao gồm đăng ký, đăng nhập và cơ chế cấp phát lại Token (Refresh Token). Đây là lớp bảo mật cốt lõi, đảm bảo rằng chỉ những người dùng hợp lệ mới có thể truy cập hệ thống.

- **Các thuộc tính (Dependencies):**
  - `userService`: Tương tác với cơ sở dữ liệu User.
  - `jwtService`: Mã hóa và giải mã Access/Refresh Token.
  - `redisService`: Lưu trữ Whitelist/Blacklist token để quản lý phiên đăng nhập.
- **Các phương thức chính:**
  - `register(registerToken, fields, passwordRaw)`: Hoàn tất đăng ký người dùng sau khi xác thực OTP thành công.
  - `login(email, passwordRaw)`: Kiểm tra thông tin đăng nhập và trả về cặp Access/Refresh Token.
  - `refreshToken(token)`: Kiểm tra tính hợp lệ của Refresh Token trong Redis và cấp phát Access Token mới (Cơ chế Silent Refresh).

  - `logout(refreshToken)`: Hủy phiên làm việc hiện tại bằng cách xóa token khỏi Redis.
  - `verifyEmail(email, otp)`: Xác minh địa chỉ email người dùng.

2. MediaService (Dịch vụ đa phương tiện) Lớp **MediaService** xử lý các tác vụ liên quan đến upload và chuyển đổi định dạng file. Đặc biệt, lớp này tích hợp quy trình transcode video sang chuẩn HLS để phục vụ streaming.

- **Các thuộc tính (Dependencies):**
  - `prismaService`: Lưu trữ metadata của file vào database.
  - `mediaQueue`: Hàng đợi (Queue) để xử lý các tác vụ nặng (transcode video) ở background worker.
  - `s3Service`: Interface giao tiếp với AWS S3 Object Storage.
- **Các phương thức chính:**
  - `handleVideoToHLS(videoFile)`: Tiếp nhận video gốc, tạo bản ghi trạng thái "Processing" và đẩy vào hàng đợi xử lý.
  - `serveVideoM3u8(id)`: Trả về file manifest (.m3u8) để client bắt đầu luồng phát video.
  - `multiUploadAndCreateMessage(files, ...)`: Xử lý upload đồng thời nhiều ảnh/file và tự động tạo tin nhắn tương ứng.
  - `handleTransformFile(files)`: Chuẩn hóa file (ví dụ: resize ảnh, convert audio) trước khi lưu trữ để tối ưu dung lượng.

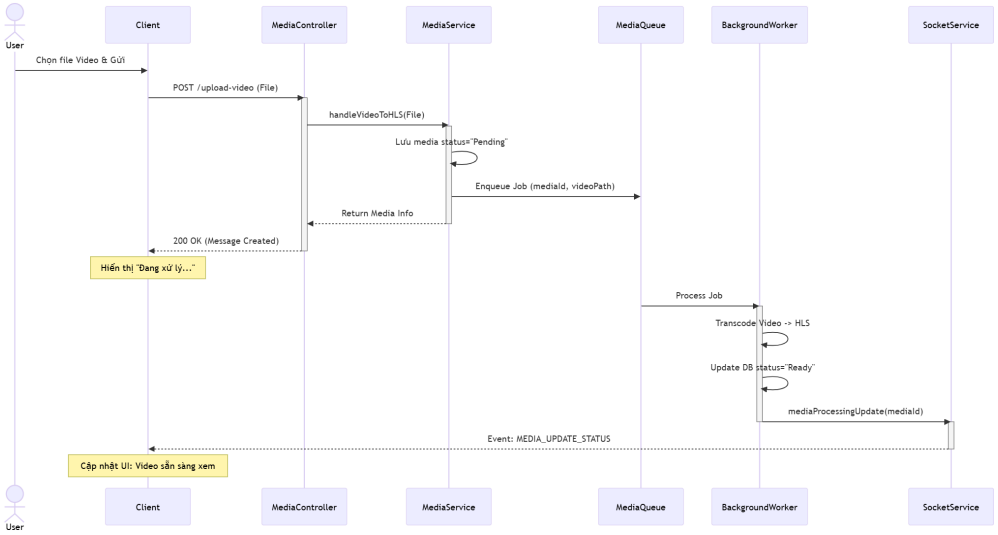
3. SocketService (Dịch vụ thời gian thực) Lớp **SocketService** đóng vai trò là tầng giao tiếp (Communication Layer), quản lý các kết nối WebSocket và phát tán sự kiện (Events) tới Client.

- **Các thuộc tính (Dependencies):**
  - `io`: Instance của Socket.io Server.
- **Các phương thức chính:**
  - `joinChat(chatId, userIds)`: Đưa người dùng vào các "Room" chat riêng biệt để nhận tin nhắn.
  - `sendMessage(chatId, payload)`: Gửi sự kiện tin nhắn mới tới tất cả thành viên trong Room.
  - `mediaProcessingUpdate(chatId, media)`: Thông báo realtime cho Client khi video đã xử lý xong (từ trang thái Processing -> Ready).
  - `memberAdded/Removed(...)`: Cập nhật danh sách thành viên trong nhóm

theo thời gian thực.

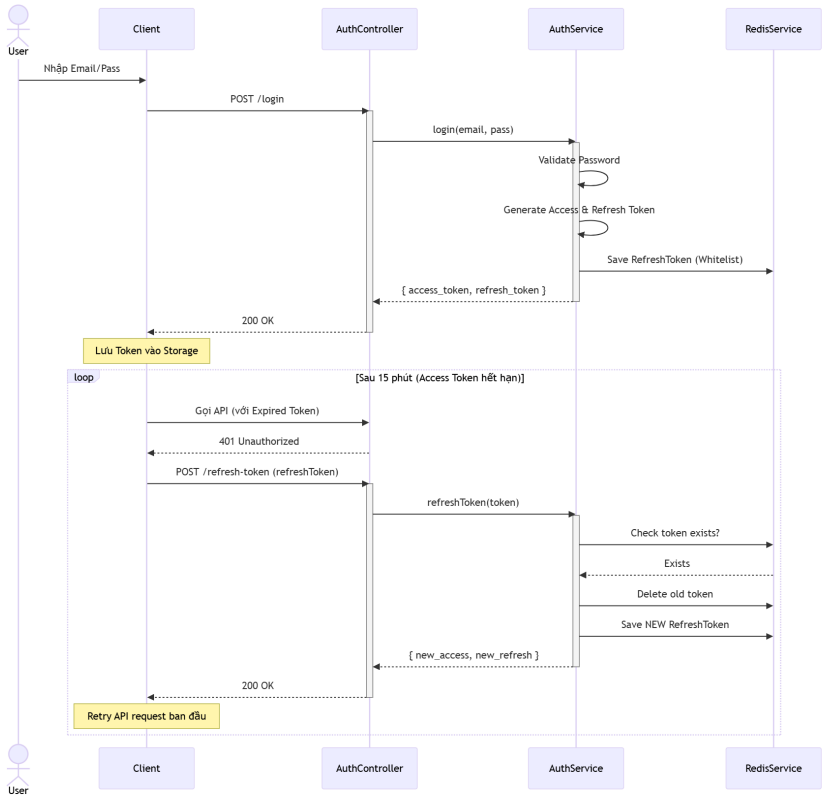
b, Biểu đồ tương tác (Sequence Diagrams)

1. Quy trình Gửi tin nhắn video và Xử lý HLS Biểu đồ dưới đây mô tả luồng dữ liệu khi người dùng gửi một video. Hệ thống tách biệt việc upload và việc xử lý video (Transcoding) để đảm bảo phản hồi nhanh cho người dùng.



Hình 4.4: Biểu đồ tuần tự use case Gửi Video và Xử lý HLS Background

2. Quy trình Đăng nhập và Tự động cấp lại Token Mô tả cơ chế bảo mật sử dụng JWT và Redis. Khi Access Token hết hạn, Client tự động yêu cầu cấp mới mà không làm gián đoạn trải nghiệm người dùng.



Hình 4.5: Biểu đồ tuần tự use case Đăng nhập và Refresh Token

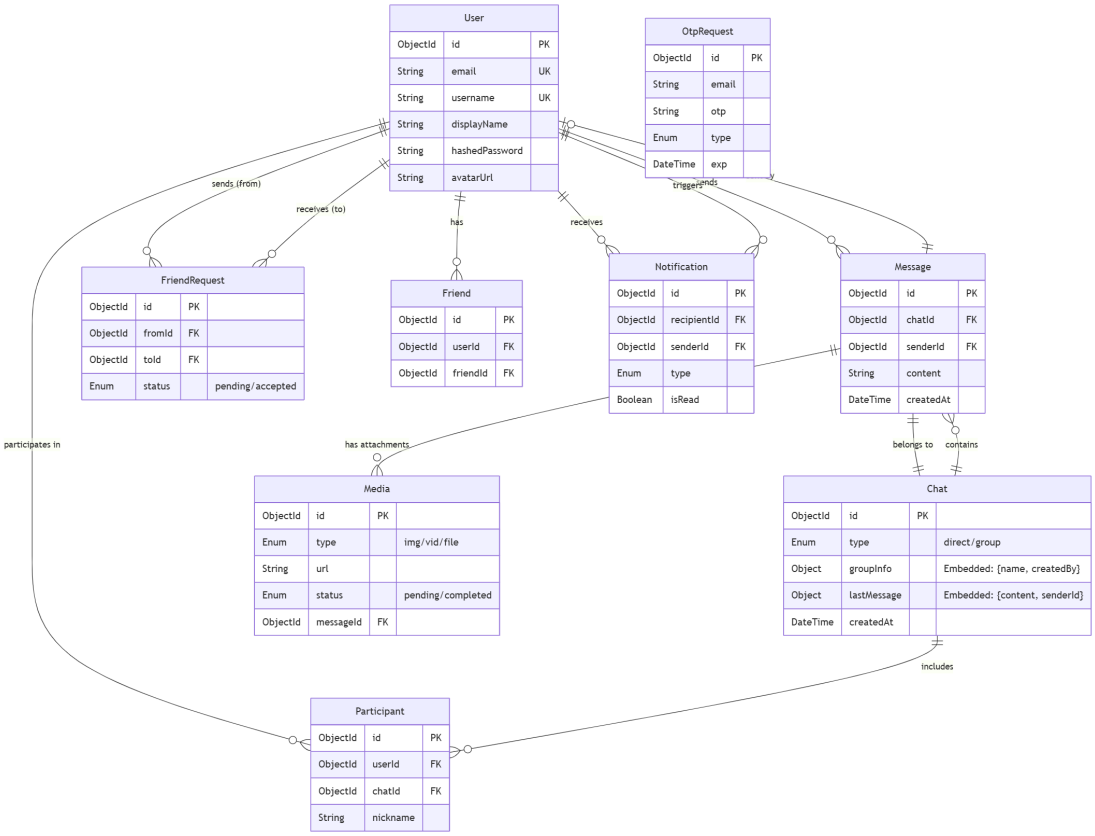
4.2.3 Thiết kế cơ sở dữ liệu

Hệ thống sử dụng hệ quản trị cơ sở dữ liệu **MongoDB** (NoSQL). Việc lựa chọn NoSQL thay vì RDBMS truyền thống (như MySQL, PostgreSQL) dựa trên các yếu tố sau:

- **High Write Throughput:** Ứng dụng chat real-time tạo ra lượng lớn tin nhắn trong thời gian ngắn, MongoDB tối ưu tốt cho việc ghi dữ liệu liên tục.
- **Flexible Schema:** Cấu trúc tin nhắn có thể thay đổi linh hoạt (text, hình ảnh, video, notification) mà không cần migrate schema phức tạp.
- **Denormalization:** Khả năng nhúng (embedding) dữ liệu giúp giảm thiểu các truy vấn JOIN đắt đỏ, tăng tốc độ đọc cho các tính năng như "Danh sách tin nhắn gần nhất".

a, Biểu đồ Quan hệ (Entity-Relationship Diagram)

Mặc dù là NoSQL, hệ thống vẫn duy trì các mối quan hệ chặt chẽ giữa các thực thể thông qua ‘ObjectId‘.



Hình 4.6: Biểu đồ quan hệ thực thể (ERD) của hệ thống

b, Chi tiết các Collections chính

Dưới đây là đặc tả chi tiết các Collection quan trọng trong cơ sở dữ liệu:

1. Collection Users Lưu trữ thông tin định danh và hồ sơ người dùng. Các trường ‘email‘ và ‘username‘ được đánh Index Unique để đảm bảo tính duy nhất.

Bảng 4.1: Cấu trúc Collection Users

Trường	**Kiểu**	**Mô tả**
_id	ObjectId	Khóa chính (Primary Key).
email	String	Email đăng nhập (Unique).
username	String	Tên định danh (Unique, Fulltext Index).
hashedPassword	String	Mật khẩu đã mã hóa (Bcrypt).
displayName	String	Tên hiển thị với người dùng khác.
avatarUrl	String	Đường dẫn ảnh đại diện.
createdAt	Date	Thời gian tạo tài khoản.

2. Collection Chats Lưu trữ thông tin về các cuộc hội thoại (Direct hoặc Group). Điểm đặc biệt là trường ‘lastMessage‘ và ‘groupInfo‘ được thiết kế dạng **Embedded Document** để tối ưu hiệu năng khi hiển thị danh sách chat.

Bảng 4.2: Cấu trúc Collection Chats

Trường	Kiểu	Mô tả
_id	ObjectId	Khóa chính.
type	Enum	Loại chat (‘direct‘ hoặc ‘group‘).
lastMessage	Object	*(Embedded)* Chứa thông tin tóm tắt tin nhắn cuối cùng (content, senderId, createdAt). Giúp client hiển thị preview mà không cần query vào bảng Message.
groupInfo	Object	*(Embedded)* Chứa tên nhóm, người tạo (chỉ có khi type=group).
participantIds	Array	Danh sách ID người tham gia (Relation).

3. Collection Messages Lưu trữ nội dung chi tiết của tin nhắn. Mỗi tin nhắn thuộc về một Chat và một Sender cụ thể.

Bảng 4.3: Cấu trúc Collection Messages

Trường	Kiểu	Mô tả
_id	ObjectId	Khóa chính.
chatId	ObjectId	Reference tới Collection Chats.
senderId	ObjectId	Reference tới Collection Users.
content	String	Nội dung văn bản của tin nhắn.
medias	Array	Danh sách các file đính kèm (Media Objects).
createdAt	Date	Thời gian gửi (Index Time-to-Live nếu cần).

4. Collection Medias Quản lý các tệp đa phương tiện. Với video, hệ thống lưu trạng thái xử lý để phục vụ quy trình transcode HLS.

Bảng 4.4: Cấu trúc Collection Medias

Trường	Kiểu	Mô tả
_id	ObjectId	Khóa chính.
type	Enum	Loại file (‘image‘, ‘video‘, ‘video_hls‘, ‘file‘).
url	String	Đường dẫn tới file trên AWS S3.
status	Enum	Trạng thái xử lý (‘pending‘, ‘processing‘, ‘completed‘, ‘failed‘).
messageId	ObjectId	Reference tới tin nhắn chứa media này.

4.3 Xây dựng ứng dụng

4.3.1 Môi trường phát triển và Thư viện

Hệ thống được phát triển trên môi trường Windows/Linux với các công cụ quản lý mã nguồn Git. Chi tiết các thư viện chính:

Bảng 4.5: Các công nghệ và thư viện cốt lõi

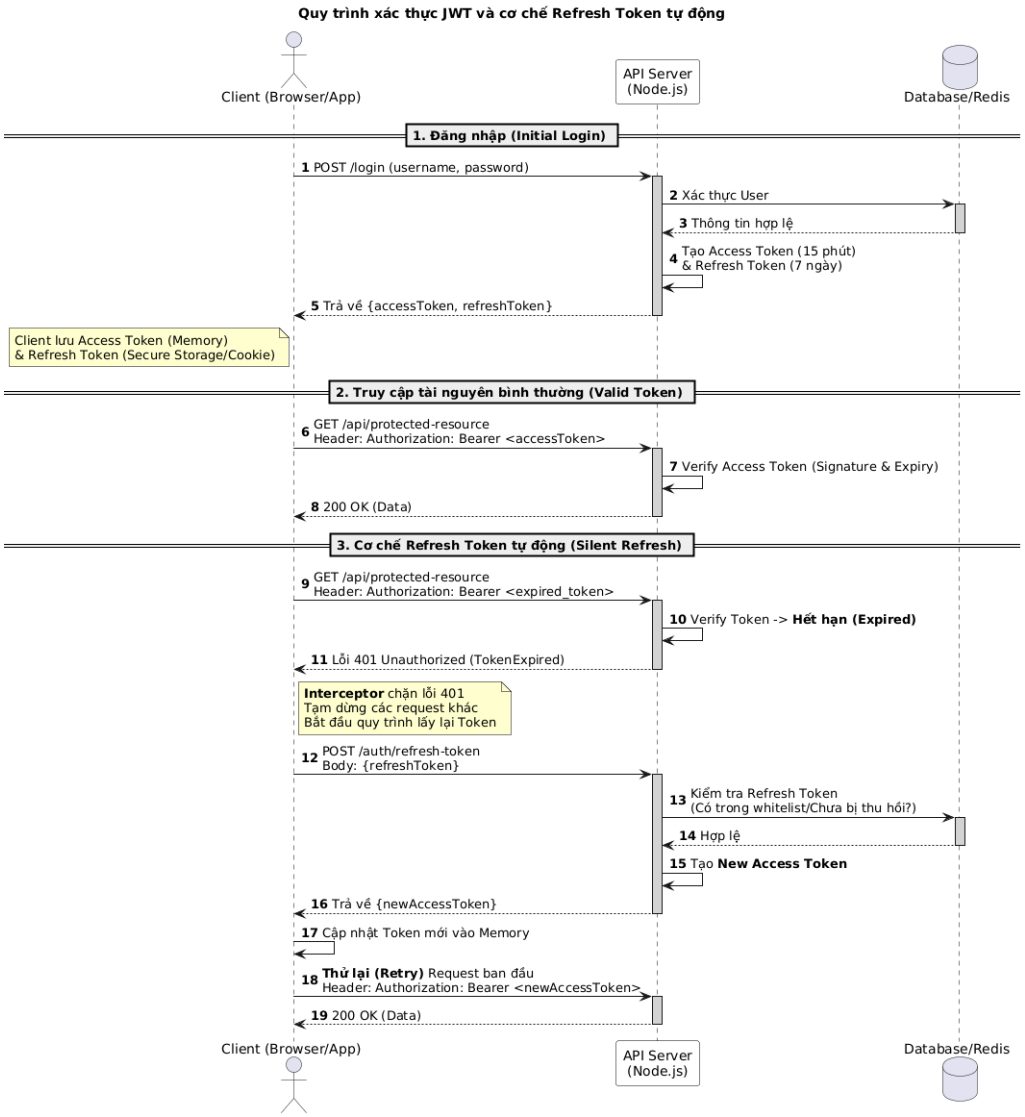
Khu vực	Công nghệ	Vai trò chính
Frontend	ReactJS 19, Vite	Framework UI và Build tool tối ưu hiệu năng.
	Zustand, React Query	Quản lý Client State và Server State (Caching).
	Socket.io-client	Giao thức thời gian thực, quản lý Room.
	Video.js / HLS.js	Player hỗ trợ phát video streaming HLS.
Backend	Node.js, Express	Nền tảng server xử lý bất đồng bộ.
	Prisma, MongoDB	ORM và Database lưu trữ document.
	WebRTC (SimplePeer)	Thư viện hỗ trợ kết nối P2P cho call video.
	Fluent-ffmpeg	Wrapper cho FFmpeg để xử lý video command.
Infra	Docker Compose	Orchestration cho môi trường dev/prod.
	AWS SDK (S3, SES)	Tích hợp dịch vụ cloud.

4.3.2 Cài đặt các quy trình nghiệp vụ chính

a, Quy trình xác thực (Authentication)

Hệ thống sử dụng cơ chế **JWT (JSON Web Token)** kép gồm Access Token (ngắn hạn, 15 phút) và Refresh Token (dài hạn, 7 ngày):

- Khi đăng nhập thành công, Server trả về cặp token.
- Access Token được dùng để xác thực API Requests.
- Khi Access Token hết hạn, Client tự động dùng Refresh Token gọi API ‘/refresh‘ để lấy token mới mà không cần user đăng nhập lại (Silent Refresh), đảm bảo trải nghiệm liền mạch.

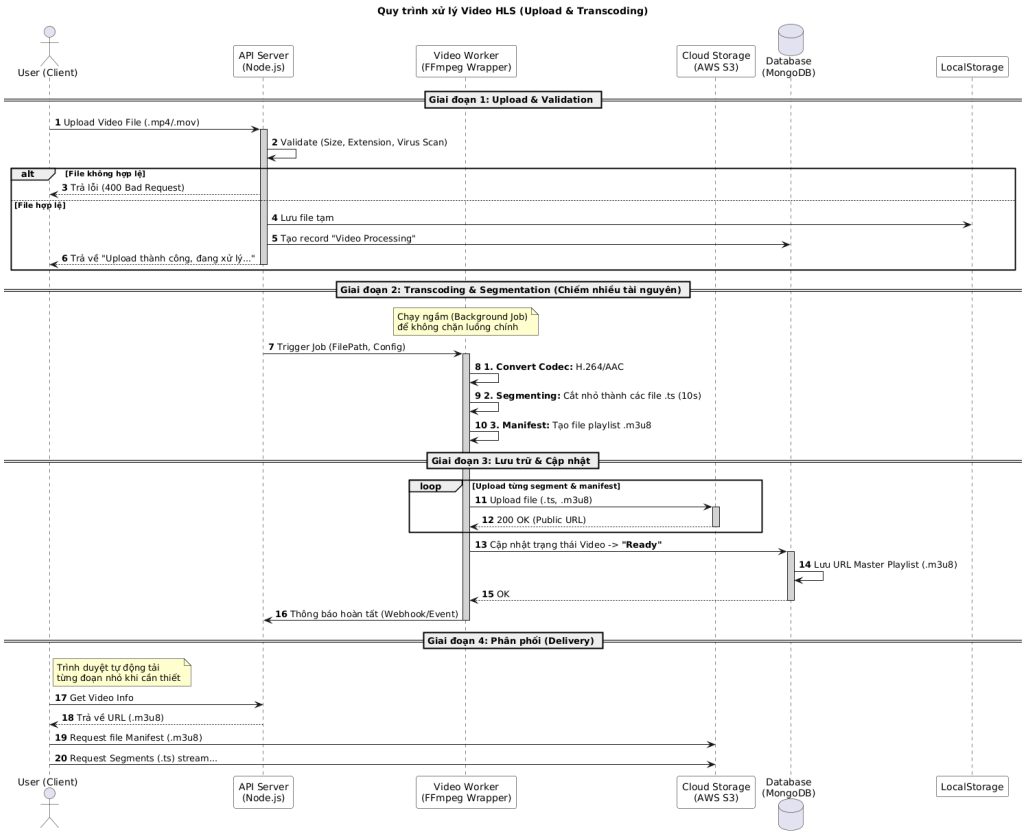


Hình 4.7: Quy trình xác thực JWT và cơ chế Refresh Token tự động

b, Quy trình xử lý Video HLS (HTTP Live Streaming)

Đây là tính năng kỹ thuật phức tạp nhất, nâng cao trải nghiệm xem video:

1. **Upload:** Client upload file video gốc (mp4, mov...) lên Server.
2. **Validation:** Server kiểm tra định dạng, kích thước và virus.
3. **Transcoding:** FFmpeg được kích hoạt, thực hiện 2 việc:
    - Convert video sang codec H.264/AAC.
    - Cắt video thành các file segments (.ts) độ dài 10s.
    - Tạo file manifest (.m3u8) chứa thông tin các segments.
4. **Storage:** Toàn bộ file segment và manifest được upload lên AWS S3.
5. **Delivery:** Client nhận link ‘.m3u8‘, trình duyệt tự động tải từng segment nhỏ để phát, cho phép tua (seek) nhanh và không cần tải toàn bộ video.

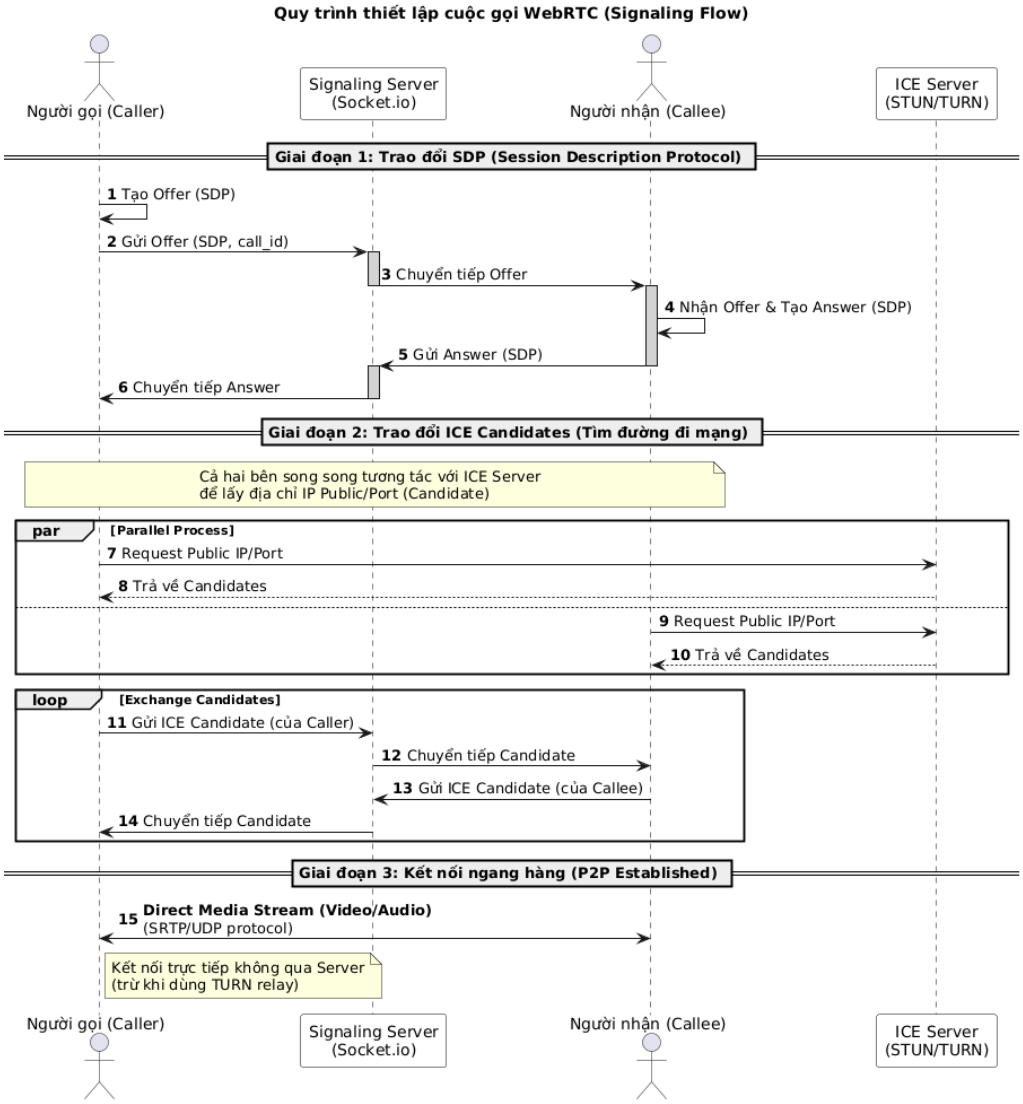


Hình 4.8: Quy trình xử lý Video HLS (HTTP Live Streaming)

c, Thiết lập cuộc gọi WebRTC

Quy trình Signaling (tín hiệu) để thiết lập kết nối P2P diễn ra qua Socket.io:

1. **Offer:** Người gọi (Caller) tạo SDP Offer và gửi qua Socket server.
2. **Answer:** Người nhận (Callee) nhận Offer, tạo SDP Answer và gửi lại.
3. **ICE Candidates:** Cả hai bên liên tục trao đổi các ứng viên mạng (IP:Port) để tìm đường đi tốt nhất (qua LAN, Wifi hoặc TURN server).
4. **Connected:** Khi đường truyền P2P thiết lập thành công, stream Video/Audio được truyền trực tiếp giữa 2 máy.

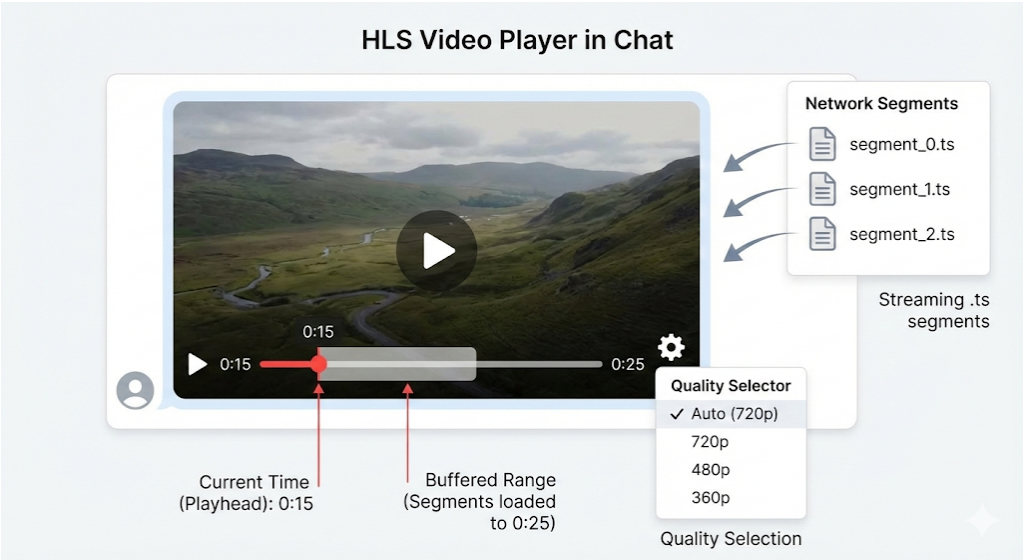


Hình 4.9: Quy trình thiết lập cuộc gọi WebRTC

4.3.3 Kết quả đạt được và Minh họa

Các module đã hoàn thiện và hoạt động ổn định:

- **Module Chat:** Hỗ trợ text, emoji, gửi ảnh, video HLS, file đính kèm. Có hiển thị "typing...", trạng thái Online/Offline.
- **Module Call:** Video Call HD, tự động chuyển đổi camera/micro, giao diện kéo thả (Draggable).
- **Module Friend:** Tìm kiếm người dùng, gửi lời mời kết bạn, chấp nhận/từ chối.



Hình 4.10: Trình phát video HLS trong khung chat, tự động buffer từng segment

4.4 Kiểm thử và Đánh giá

4.4.1 Phương pháp kiểm thử

Nhóm thực hiện kiểm thử theo cấp độ **Integration Testing** (Kiểm thử tích hợp) và **System Testing** (Kiểm thử hệ thống).

- **API Testing:** Sử dụng Postman để kiểm tra 100% các endpoints, đảm bảo Status Code và Response Body chuẩn xác cho các trường hợp Success/Error.
- **Socket Testing:** Giả lập nhiều Client kết nối cùng lúc để test độ trễ của tin nhắn và tính đồng bộ của trạng thái ("Đã xem").

4.4.2 Kết quả thực nghiệm

- **Độ trễ tin nhắn:** < 100ms trong điều kiện mạng 4G tiêu chuẩn.
- **Xử lý Video:** Video 100MB được transcode và sẵn sàng phát sau khoảng 15-20 giây.
- **Hiệu năng WebRTC:** Duy trì FPS 24-30 ổn định. Khi Packet Loss > 5%, chất lượng video tự động giảm bitrate để ưu tiên âm thanh (Audio-first).

4.5 Triển khai (Deployment)

4.5.1 Mô hình triển khai Docker Compose

Quy trình triển khai được tự động hóa hoàn toàn bằng file ‘docker-compose.prod.yml‘. Một câu lệnh ‘docker-compose up -d‘ sẽ khởi tạo toàn bộ hạ tầng:

1. **Service Database:** MongoDB khởi tạo với Replica Set (mô phỏng) để hỗ trợ Transaction.
2. **Service Redis:** Redis khởi động và load cấu hình persistence.
3. **Service Backend:** Node.js app chờ DB/Redis sẵn sàng (healthcheck) rồi mới

start. Biến môi trường được nạp từ file ‘.env‘ bảo mật.

4. **Service Proxy:** Nginx container mount volume chứng chỉ SSL (Let’s Encrypt), cấu hình chặn các request không hợp lệ và cache file tĩnh.

4.5.2 Cấu hình Nginx và Bảo mật

Nginx đóng vai trò "người gác cổng" quan trọng:

- **SSL Termination:** Mã hóa toàn bộ traffic bằng HTTPS.
- **Client Max Body Size:** Cấu hình giới hạn upload (ví dụ 50MB) để chống tấn công DoS qua upload file.
- **Header Security:** Ẩn thông tin Server (X-Powered-By), cấu hình CORS chặt chẽ chỉ cho phép domain của Client truy cập API.

CHƯƠNG 5. CÁC GIẢI PHÁP VÀ ĐÓNG GÓP NỔI BẬT

Chương này trình bày các giải pháp kỹ thuật, đóng góp cốt lõi và những vấn đề thách thức mà đồ án đã giải quyết được trong quá trình nghiên cứu và phát triển hệ thống nhắn tin, gọi điện trực tuyến. Các giải pháp này tập trung vào việc tối ưu hóa hiệu năng, đảm bảo tính thời gian thực (real-time) và nâng cao trải nghiệm người dùng cuối (User Experience).

5.1 Giải pháp tối ưu hóa truyền tải Video bằng giao thức HLS

5.1.1 Đặt vấn đề

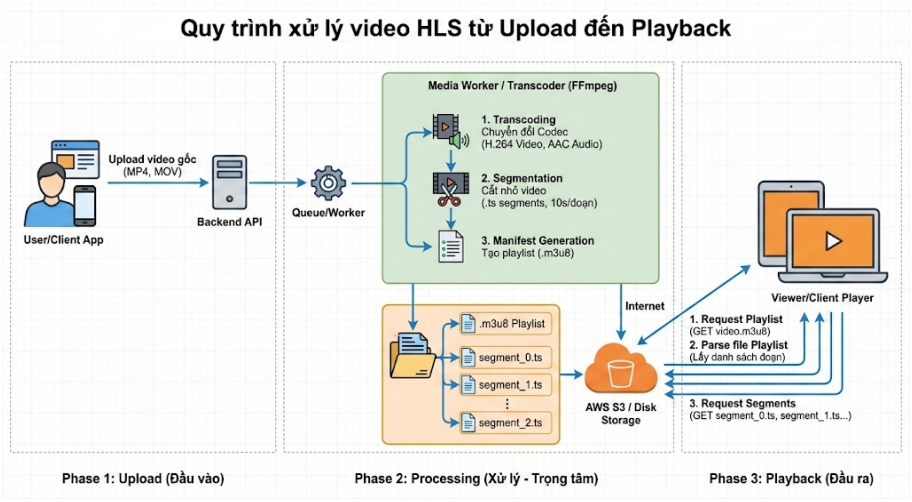
Trong các ứng dụng nhắn tin đa phương tiện hiện nay, nhu cầu chia sẻ và xem video là rất lớn. Tuy nhiên, cách tiếp cận truyền thống (HTTP Progressive Download) bộc lộ nhiều hạn chế nghiêm trọng. Khi người dùng gửi một video dung lượng lớn (ví dụ: file MP4 100MB):

- **Độ trễ khi phát (Startup Latency):** Trình duyệt phải tải xuống một phần đáng kể của file (header/metadata) trước khi có thể bắt đầu phát, gây ra tình trạng chờ đợi khó chịu.
- **Lãng phí băng thông:** Nếu người dùng chỉ xem 10 giây đầu của video dài 10 phút, phương pháp cũ vẫn có thể đã tải trước (buffer) hàng chục MB dữ liệu không cần thiết.
- **Khó khăn khi tua (Seeking):** Việc tua đến một thời điểm chưa được tải về thường gây ra độ trễ lớn do client phải gửi request range mới và chờ server phản hồi.

5.1.2 Giải pháp đề xuất

Để giải quyết triệt để các vấn đề trên, đồ án đề xuất và triển khai quy trình xử lý video dựa trên giao thức **HLS (HTTP Live Streaming)** kết hợp với thư viện xử lý FFmpeg và hạ tầng lưu trữ AWS S3. Kiến trúc giải pháp bao gồm các bước sau:

1. **Transcoding & Segmentation (Phía Server):** Thay vì lưu trữ file gốc, hệ thống sử dụng FFmpeg để chuyển mã (transcode) video sang định dạng chuẩn H.264/AAC. Quan trọng hơn, video được chia cắt (segment) thành các file ‘.ts‘ nhỏ (time-based segmentation), mỗi file có độ dài khoảng 10 giây.
2. **Tạo Manifest File:** FFmpeg đồng thời tạo ra một file chỉ mục ‘.m3u8‘ (playlist manifest). File này chứa danh sách các đường dẫn đến từng file ‘.ts‘ và thông tin về thời lượng, độ phân giải.
3. **Adaptive Streaming (Phía Client):** Trình duyệt phát video không tải toàn bộ file một lúc. Thay vào đó, nó tải file manifest ‘.m3u8‘ trước (chỉ vài KB), sau đó dựa trên thời điểm người dùng muốn xem để tải chính xác các segment ‘.ts‘ tương ứng.



Hình 5.1: Quy trình xử lý video HLS từ Upload đến Playback

5.1.3 Kết quả đạt được

Việc áp dụng HLS đã mang lại hiệu quả rõ rệt:

- **Giảm 90% độ trễ khởi tạo:** Video bắt đầu phát gần như ngay lập tức sau khi nhấn play, do player chỉ cần tải file manifest nhẹ và segment đầu tiên (khoảng 1-2MB).
- **Tiết kiệm 40-60% băng thông:** Hệ thống chỉ tải dữ liệu tại thời điểm người dùng đang xem. Nếu người dùng tắt video giữa chừng, không có dữ liệu dư thừa nào bị tải về.
- **Trải nghiệm Seeking mượt mà:** Người dùng có thể tua đến bất kỳ vị trí nào trong video ngay lập tức mà không cần chờ tải file từ đầu.

5.2 Xây dựng kiến trúc giao tiếp thời gian thực tin cậy

5.2.1 Đặt vấn đề

Thách thức lớn nhất của các ứng dụng Real-time Communication (RTC) là duy trì trạng thái đồng bộ giữa các Client và Server trong môi trường mạng không ổn định.

- Làm sao để quản lý hàng nghìn kết nối WebSocket đồng thời mà không làm quá tải server?
- Làm thế nào để đảm bảo tính năng gọi điện (WebRTC) và nhắn tin (Socket.io) hoạt động song song mà không xung đột?
- Vấn đề mất kết nối (disconnect) và kết nối lại (reconnect) xử lý thế nào để không mất tin nhắn?

5.2.2 Giải pháp đề xuất

Đồ án xây dựng một kiến trúc giao tiếp phân tầng (Layered Communication Architecture) để giải quyết các vấn đề trên:

a, Cơ chế xác thực và quản lý Socket (Socket Authentication Guard)

Mọi kết nối WebSocket đều phải trải qua bước xác thực nghiêm ngặt bằng JWT ngay tại giai đoạn Handshake. Middleware ‘SocketAuthMiddleware‘ sẽ:

- Xác minh tính hợp lệ của Token.
- Gán ‘userId‘ vào socket instance.
- Tự động `join` socket vào các `Room` riêng tư (ví dụ: `room:user_id`) để nhận thông báo cá nhân.

b, Tách biệt luồng Signaling và Data

Đối với tính năng gọi video, hệ thống tách biệt luồng tín hiệu (Signaling) và luồng dữ liệu media (Media Stream):

- **Signaling Channel (Socket.io):** Chỉ dùng để trao đổi các gói tin SDP (Session Description Protocol) và ICE Candidates dung lượng nhỏ. Server đóng vai trò trung chuyển (Relay) tin cậy.
- **Media Channel (WebRTC P2P):** Sau khi bắt tay thành công, dữ liệu hình ảnh/âm thanh đi trực tiếp giữa 2 máy peer-to-peer, giảm tải hoàn toàn cho Server và giảm độ trễ xuống mức thấp nhất (< 200ms).

c, Cơ chế Redis Adapter cho Horizontal Scaling

Để hệ thống có khả năng mở rộng, trạng thái của các socket không được lưu trong bộ nhớ RAM của một server đơn lẻ. Đồ án sử dụng **Redis Adapter** kết hợp với Socket.io. Khi cần gửi tin nhắn cho User A, server sẽ publish một sự kiện vào Redis, Redis sẽ phân phối sự kiện đó đến đúng server Instance mà User A đang kết nối. Giải pháp này cho phép hệ thống mở rộng lên nhiều server con (Cluster) mà không làm gián đoạn luồng giao tiếp.

5.2.3 Kết quả đạt được

- **Độ tin cậy cao:** Hệ thống duy trì kết nối ổn định ngay cả khi chuyển đổi mạng (Wifi sang 4G). Cơ chế tự động Reconnect của Socket.io hoạt động hiệu quả.
- **Khả năng mở rộng:** Kiến trúc Redis Adapter cho phép dễ dàng thêm server mới để chịu tải khi lượng người dùng tăng đột biến.
- **Trải nghiệm liền mạch:** Người dùng có thể vừa gọi điện video vừa nhắn tin văn bản/hình ảnh mà không gặp tình trạng giật lag hay mất đồng bộ.

5.3 Quy trình triển khai tự động hóa và đóng gói (Containerization)

5.3.1 Đặt vấn đề

Một vấn đề kinh điển trong phát triển phần mềm là sự không nhất quán giữa môi trường phát triển (Development) và môi trường thực tế (Production).

- Lỗi "Works on my machine": Code chạy tốt trên máy Dev nhưng lỗi trên Server do khác phiên bản Node.js, OS hoặc cấu hình thư viện.

- Quá trình cài đặt môi trường (Setup Environment) tốn nhiều thời gian và dễ sai sót khi phải cấu hình thủ công từng dịch vụ (DB, Redis, Nginx...).

5.3.2 Giải pháp đề xuất

Đồ án áp dụng triệt để công nghệ **Containerization (Docker)** để đóng gói ứng dụng:

- **Dockerize từng Service:** Mỗi thành phần (Backend API, Frontend App, Worker) được đóng gói trong một Docker Image riêng biệt với file ‘Dockerfile‘ tối ưu (sử dụng Multi-stage build để giảm dung lượng image).
- **Orchestration với Docker Compose:** File ‘docker-compose.yml‘ đóng vai trò là "bản thiết kế" hạ tầng, định nghĩa toàn bộ các service, network, volume và biến môi trường cần thiết.
- **Nginx Reverse Proxy:** Sử dụng Nginx container làm cổng giao tiếp duy nhất ra Internet, che giấu kiến trúc bên trong và tăng cường bảo mật.
- **Triển khai Kubernetes (K8s) (Đề xuất):** Để đáp ứng nhu cầu mở rộng quy mô lớn trong tương lai, đồ án đề xuất lộ trình nâng cấp từ Docker Compose lên Kubernetes (K8s). K8s sẽ cung cấp khả năng tự động Scaling (Auto-scaling Pods) dựa trên tải CPU/RAM và cơ chế Self-healing (tự khởi động lại container khi lỗi), đảm bảo tính sẵn sàng cao (High Availability) cho hệ thống.

5.3.3 Kết quả đạt được

- **Triển khai "One-Click":** Việc khởi chạy toàn bộ hệ thống phức tạp chỉ tốn đúng 1 câu lệnh (‘docker-compose up‘), giúp giảm thời gian setup server từ vài giờ xuống còn vài phút.
- **Môi trường nhất quán:** Đảm bảo 100% sự tương thích giữa Local và Server, loại bỏ hoàn toàn các lỗi liên quan đến môi trường.
- **Dễ dàng bảo trì/nâng cấp:** Việc cập nhật version của một service (ví dụ nâng cấp Node.js 18 lên 20) chỉ cần sửa 1 dòng trong Dockerfile và rebuild, không ảnh hưởng đến các service khác.

CHƯƠNG 6. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

6.1 Kết luận

Trong khuôn khổ của đồ án môn học nghiên cứu tốt nghiệp 2, tôi đã tập trung nghiên cứu và xây dựng thành công **"Hệ thống nhắn tin và gọi điện trực tuyến thời gian thực"** với đầy đủ các tính năng của một mạng xã hội hiện đại. Sản phẩm không chỉ dừng lại ở mức độ demo các tính năng cơ bản mà còn đi sâu giải quyết các bài toán kỹ thuật thách thức về truyền tải đa phương tiện và tối ưu hóa hệ thống.

Các kết quả chính đã đạt được bao gồm:

- **Về mặt kiến trúc:** Đã xây dựng thành công **Kiến trúc Client-Server** với mô hình tách biệt rõ ràng giữa Frontend (Client) và Backend (Server). Việc tổ chức mã nguồn theo hướng Module hóa và sử dụng Docker để đóng gói các dịch vụ giúp việc triển khai, bảo trì và mở rộng hệ thống trở nên dễ dàng và linh hoạt hơn.
- **Về mặt công nghệ:** Đã làm chủ và ứng dụng thành công các công nghệ tiên tiến:
  - **WebRTC:** Cho phép gọi điện video P2P chất lượng cao với độ trễ thấp (< 200ms).
  - **HLS Streaming:** Giải quyết triệt để vấn đề xem video trực tuyến, giúp video tải nhanh và mượt mà ngay cả trong điều kiện mạng kém.
  - **Docker & DevOps:** Tự động hóa quy trình triển khai, đảm bảo tính nhất quán của môi trường sản phẩm.
- **Về mặt sản phẩm:** Ứng dụng hoạt động ổn định, giao diện thân thiện, hỗ trợ đa nền tảng (Responsive Design) và đầy đủ các tính năng từ xác thực, kết bạn đến nhắn tin đa phương tiện.

So sánh với các sản phẩm tương tự trên thị trường (như Zalo Web, Messenger Web), đồ án tuy chưa thể sánh bằng về bề dày tính năng và hệ sinh thái, nhưng đã tiệm cận được về mặt trải nghiệm người dùng cốt lõi (tốc độ gửi tin, chất lượng cuộc gọi) và đặc biệt là cơ chế xử lý video streaming tối ưu mà nhiều dự án sinh viên thường bỏ qua.

6.2 Hạn chế

Bên cạnh những kết quả đạt được, đồ án vẫn còn tồn tại một số hạn chế nhất định do giới hạn về thời gian và nguồn lực:

- **Giới hạn của mô hình WebRTC Mesh:** Hiện tại chức năng gọi nhóm (Group Call) đang sử dụng kiến trúc Mesh (kết nối đa điểm P2P). Kiến trúc này đơn giản nhưng tiêu tốn nhiều băng thông của Client khi số lượng người trong phòng tăng lên (hiệu năng giảm sút rõ rệt nếu > 4 người).
- **Chưa có ứng dụng di động (Native Mobile App):** Hệ thống mới chỉ hoạt động trên nền tảng Web. Mặc dù giao diện đã Responsive nhưng trải nghiệm trên trình

duyệt mobile vẫn chưa thể mượt mà và tận dụng tốt phần cứng như Native App.

- **Vấn đề bảo mật nâng cao:** Hệ thống đã có HTTPS và JWT, nhưng chưa triển khai mã hóa đầu cuối (End-to-End Encryption - E2EE) cho toàn bộ nội dung tin nhắn, đây là tiêu chuẩn quan trọng của các ứng dụng chat hiện đại.

6.3 Hướng phát triển

Dựa trên nền tảng hiện có, đồ án đề xuất các hướng phát triển tiếp theo để hoàn thiện và nâng cấp hệ thống:

- **Nâng cấp kiến trúc WebRTC với SFU:** Chuyển đổi từ mô hình Mesh sang sử dụng **SFU (Selective Forwarding Unit)** bằng cách tích hợp Media Server (như Mediasoup hoặc Kurento). Điều này sẽ giảm tải băng thông cho Client và cho phép tổ chức các cuộc họp trực tuyến quy mô lớn (trăm người).
- **Phát triển Mobile App với React Native:** Tận dụng mã nguồn ReactJS hiện có và kiến trúc API Backend để xây dựng phiên bản Mobile App (iOS/Android), hỗ trợ Push Notification và tích hợp sâu với danh bạ điện thoại.

TÀI LIỆU THAM KHẢO

[1] *Node.js*, https://nodejs.org/, Truy cập ngày: 14/01/2026.

[2] *Prisma orm*, https://www.prisma.io/, Truy cập ngày: 14/01/2026.

[3] *Webrtc: Real-time communication*, https://webrtc.org/, Truy cập ngày: 14/01/2026.

[4] *Mongodb: The developer data platform*, https://www.mongodb.com/, Truy cập ngày: 14/01/2026.

[5] *Amazon s3 - simple storage service*, https://aws.amazon.com/s3/, Truy cập ngày: 14/01/2026.

[6] *Docker: Accelerated container application development*, https://www.docker.com/, Truy cập ngày: 14/01/2026.

[7] *Nginx: Advanced load balancer, web server, & reverse proxy*, https://www.nginx.com/, Truy cập ngày: 14/01/2026.

[8] *Amazon simple email service (ses)*, https://aws.amazon.com/ses/, Truy cập ngày: 14/01/2026.