

BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC MỎ - ĐỊA CHẤT



NGHIÊN CỨU KHOA HỌC SINH VIÊN

**XÂY DỰNG HỆ THỐNG CHAT BẢO MẬT ĐA NGƯỜI DÙNG SỬ DỤNG JAVA
SWING VÀ TCP SOCKET VỚI MÃ HÓA AES**

SINH VIÊN THỰC HIỆN

Trần Công Nam
Phạm Tiến Chính
Phạm Thanh Sơn
Phạm Minh Chiến

CÁN BỘ HƯỚNG DẪN

GVC. ThS. Nguyễn Tuấn Anh

Bộ môn: Mạng Máy Tính

:

HÀ NỘI- 05/2026

Mục lục

Lời cảm ơn	7
Lời mở đầu.....	8
TỪ ĐIỂN THUẬT NGỮ (GLOSSARY).....	9
1. Các thuật ngữ về An toàn thông tin và Mã hóa	9
2. Hạ tầng và Giao thức truyền thông đa phương tiện	9
3. Trí tuệ nhân tạo cục bộ (Local AI).....	10
4. Kiến trúc và Giao diện ứng dụng.....	10
5. Bảng tổng hợp đối sánh và Luồng kiến trúc kỹ thuật	10
Chương 1 Giới thiệu đề tài và Bối cảnh	12
1.1 Đặt vấn đề và Bối cảnh thực tiễn.....	12
Phân tích giải pháp công nghệ và Kiến trúc hệ thống.....	13
Mục tiêu và Đóng góp của đề tài.....	16
1.2. Nhu cầu tích hợp AI (Trợ lý ảo) và giao tiếp đa phương tiện an toàn.....	16
1. Nguy cơ an toàn thông tin khi tích hợp Trợ lý ảo AI dựa trên Điện toán đám mây.....	16
2. Thách thức bảo mật trong truyền thông đa phương tiện thời gian thực (Video/Audio Call)	17
3. Sơ đồ Luồng dữ liệu và Ranh giới bảo mật hệ thống	17
4. Ma trận phân tích sự cần thiết kỹ thuật và Giải pháp đối sánh	18
1.3 Phân tích nền tảng Telegram, Signal	19
Phân tích nền tảng Telegram, Signal.....	19
Chương 2: Cơ sở lý thuyết và công nghệ sử dụng	21
2.1 Java & Java Swing (Phát triển giao diện Client với FlatLaf)	21
2.2 Môi trường thực thi (JDK 17+, JRE).....	22
1. Cơ chế quản lý bộ nhớ và Bộ thu gom rác (Garbage Collection - ZGC & G1GC).....	22
2. Các tính năng ngôn ngữ và Tối ưu hóa kiến trúc hạ tầng	22
3. Kiến trúc tiền xử lý mật mã học qua JCA/JCE trên JDK 17	23
4. Bảng thông số cấu hình và Yêu cầu môi trường tối thiểu	24
2.3. AES-256 (Mã hóa đối xứng cho nội dung tin nhắn).....	24
Kiến trúc Containerization với Docker & Docker-compose	25
2.4. Nền tảng Jitsi Meet Stack (Prosody, Jicofo, JVB) cho Video Call.....	29
1. Kiến trúc phân rã các thành phần trong Jitsi Meet Stack	29
2. Phân tích luồng thiết lập cuộc gọi và Định tuyến WebRTC.....	29

3. Cơ chế tối ưu hóa băng thông bằng giải thuật Simulcast và BWE	30
4. Bảng thông số cấu hình hệ thống Jitsi Stack trong môi trường Docker.....	30
2.3.1. Jitsi Meet Stack và Kiến trúc Containerized	32
Tổng quan về Local AI và Ollama Framework.....	32
2.5 Cơ chế tích hợp Ollama API vào hệ thống Chat (Local LLM prompt engineering).....	35
1. Kiến trúc kết nối không đồng bộ và Kỹ thuật truyền dữ liệu dạng dòng (Streaming API)	35
2. Kỹ nghệ định hướng cấu trúc (Prompt Engineering) và Kiểm soát định dạng JSON	36
3. Các tham số điều khiển siêu dữ liệu suy luận (Inference Hyperparameters)	37
2.6 Local AI & LLM (Mô hình ngôn ngữ lớn chạy offline).....	38
2.7 Framework Ollama (Cách tích hợp Llama 3 / Mistral vào Java)	38
2.8. Mã hóa đầu cuối (End-to-End Encryption - E2EE) hoạt động như thế nào?.....	39
Quy trình thực thi thông điệp mã hóa đầu cuối (E2EE).....	40
Tiến trình thực thi thông điệp mã hóa đầu cuối (E2EE)	40
2.9. Quản lý Session và Thuật toán mã hóa khóa (Sử dụng AES-256 và RSA-2048 trong Java)	43
1. Thiết lập cặp khóa RSA-2048 và Cơ chế bảo vệ Private Key nội bộ.....	43
2. Cơ chế quản lý Phiên (Session Key) và Luồng giao tiếp lai (Hybrid Encryption)	44
3. Hiện thực thuật toán mã hóa AES-256 ở chế độ GCM trong Java	45
Chương 3: ĐẶC TẢ YÊU CẦU PHẦN MỀM (SRS) và THIẾT KẾ HỆ THỐNG (SDD & OOD)	45
3.1. Danh sách tác nhân (Actors trong hệ thống)	45
Phân tích đặc điểm tác nhân và Ranh giới hệ thống.....	45
Đặc tả chi tiết các Tác nhân trong hệ thống.....	46
Sơ đồ mối quan hệ và Ranh giới tương tác của Tác nhân	48
3.2. Biểu đồ Use Case tổng quát.....	49
1. Xác định các Tác nhân (Actors) trong hệ thống.....	49
2. Biểu đồ Use Case tổng quát hệ thống	50
3. Phân tích phân rã kiến trúc kỹ thuật của các Use Case chính	50
4. Bảng đặc tả chi tiết dòng sự kiện của các Use Case cốt lõi.....	50
3.3.1. Đặc tả yêu cầu: Phân hệ Xác thực & Quản lý Phiên làm việc (Kèm sơ đồ Sequence) ..	51
Phân tích Yêu cầu Kỹ thuật: Nhóm chức năng Xác thực & Phiên làm việc	51
Đặc tả yêu cầu: Nhóm chức năng Xác thực & Phiên làm việc.....	51
3.3.2. Đặc tả yêu cầu: Nhóm chức năng Nhắn tin & Trạng thái Real-time	53

3.3.3. Đặc tả yêu cầu: Nhóm chức năng Giao tiếp đa phương tiện (Video call)	56
1. Phân tích yêu cầu chức năng và Phi chức năng chuyên sâu	56
2. Biểu đồ luồng dữ liệu logic và Trạng thái kết nối cuộc gọi	56
3. Kiến trúc kỹ thuật và Cơ chế mã hóa bảo mật truyền thông (DTLS-SRTP)	58
4. Bảng đặc tả yêu cầu chi tiết của các Use Case thành phần	58
3.3.4. Đặc tả yêu cầu: Nhóm chức năng Truyền tải dữ liệu an toàn	60
3.3.5. Đặc tả yêu cầu: Nhóm chức năng Tương tác AI (Ollama Chatbot)	62
1. Phân tích yêu cầu chức năng và Phi chức năng chuyên sâu	63
2. Biểu đồ luồng trạng thái xử lý Prompt và Phân tách dữ liệu	63
3. Kiến trúc tiền xử lý ngữ cảnh an toàn và Kỹ thuật điều phối luồng (Multi-threading)....	65
4. Bảng đặc tả yêu cầu chi tiết của các Use Case phân hệ AI	65
Luồng sự kiện chính (Tiếp tục)	66
2. Hệ thống chuyển đổi giao diện sang trạng thái chờ, khởi tạo một Worker Thread riêng biệt để xử lý.	66
3. Hệ thống Client đóng gói cấu trúc dữ liệu gồm: System Prompt định hình vai trò, mảng chứa ngữ cảnh lịch sử đối thoại trượt, và nội dung câu hỏi mới.	66
4. Thực hiện lệnh HTTP POST không đồng bộ tới địa chỉ <code>http://localhost:11434/api/chat</code> với tham số cấu hình <code>"stream": true</code>	66
5. Khi nhận được từng khối dữ liệu (Data chunks) chứa cấu trúc JSON từ luồng mạng, hệ thống phân tách chuỗi Token và sử dụng <code>SwingUtilities.invokeLater()</code> để cập nhật text liên tục vào bong bóng chat.	66
Luồng ngoại lệ.....	66
Trường hợp lỗi kết nối mạng máy chủ AI (Mất kết nối Container):.....	66
Nếu quá trình kết nối ném ra ngoại lệ <code>ConnectException</code> hoặc mã trạng thái HTTP trả về khác 200, Worker Thread sẽ phát tín hiệu lỗi về EDT. Giao diện Swing dừng hiệu ứng loading, hiển thị bong bóng chat màu đỏ với nội dung thông báo lỗi: <i>“Không thể kết nối đến máy chủ AI cục bộ. Vui lòng kiểm tra lại hạ tầng Docker.”</i>	66
Hậu điều kiện.....	66
Toàn bộ câu trả lời hoàn chỉnh được kết xuất thành công; nội dung phản hồi được lưu vết vào cơ sở dữ liệu MySQL cục bộ phục vụ cho các phiên làm việc tiếp theo.	66
3.4. Thiết kế Hướng đối tượng (Class Diagram cho các module View, Model, Controller) ...	66
1. Kiến trúc phân tầng cấu trúc cấu thành hệ thống	67
2. Biểu đồ lớp hệ thống (System Class Diagram)	67
3. Phân tích cơ chế tương tác và Các mẫu thiết kế (Design Patterns).....	68
4. Đặc tả giao diện lập trình các lớp lỗi (Class Interface Specifications)	68

3.5. Thiết kế Dữ liệu (Database Schema cho Users, Messages - Kèm bảng mô tả các trường DB)	69
Đặc tả cấu trúc chi tiết các Bảng dữ liệu cốt lõi	70
Mô hình Quan hệ và Liên kết Thực thể (Entity-Relationship Diagram)	72
3.6. Thiết kế Giao diện người dùng (UI/UX - Modern Aesthetic, Glassmorphism)	74
1. Hiện thực hóa Hiệu ứng Kính mờ (Glassmorphism) trên Java Swing	74
2. Kiến trúc Render Bong bóng Chat Gradient và Chế độ tối (Dark Mode)	74
2. Kiến trúc Render Bong bóng Chat Gradient và Chế độ tối (Dark Mode)	74
3. Sơ đồ Kiến trúc phân tầng đồ họa và Luồng xử lý sự kiện UI.....	75
4. Các thông số Kỹ thuật và Ma trận chỉ số Thiết kế UI/UX	75
5. Thiết kế luồng UX đặc thù cho các tính năng Bảo mật và Trợ lý AI	77
CHƯƠNG 4: THIẾT KẾ GIAO DIỆN NGƯỜI DÙNG (UI/UX).....	77
4.1. Nguyên tắc thiết kế (Design Principles).....	77
4.2. Phân hệ Xác thực và Quản lý Tài khoản (Auth & Profiles)	78
4.3. Phân hệ Nhắn tin và Tương tác cốt lõi (Core Messaging & AI)	86
4.4. Phân hệ Tiện ích Chat và Quản lý Cuộc trò chuyện	91
4.5. Phân hệ Giao tiếp Đa phương tiện (Jitsi Video Call).....	98
4.6. Phân hệ Mạng xã hội nội bộ và Cộng tác	105
4.7. Phân hệ Cài đặt và Tùy chỉnh (Settings & Configs)	110
4.8. Phân hệ Quản trị Admin và Bảo mật Hệ thống	112
CHƯƠNG 5: CÀI ĐẶT VÀ KIỂM THỬ (IMPLEMENTATION & TESTING)	118
5.1.1. Thiết lập file docker-compose.yml (Giải thích các service: ollama, mysql, jitsi)	118
1. Cấu trúc kiến trúc phân tầng mạng và Phân rã Dịch vụ (Services Decomposition).....	118
2. Giải thích chi tiết các Service cấu hình học thuật.....	119
3. Bản đặc tả tham số cấu hình tệp tin docker-compose.yml	120
4. Cơ chế đồng bộ khởi chạy và Kiểm soát phụ thuộc (Service Dependency)	120
5. Cơ chế đồng bộ khởi chạy và Kiểm soát phụ thuộc (Service Dependency)	121
5.1.2. Cấu hình biên dịch (java -cp với thư viện ngoài)	121
a) Kiến trúc phân rã thư viện phụ thuộc (Dependencies)	122
b) Quy trình và sơ đồ luồng biên dịch thủ công qua CLI	122
c) Đặc tả tập lệnh biên dịch và thực thi hệ thống.....	123
d) Phân tích cơ chế nạp lớp (Class Loading Mechanism) và xử lý lỗi hệ thống.....	124
5.3. Kế hoạch và Chiến lược kiểm thử (Test Plan).....	124

CHƯƠNG 6: TỔNG KẾT VÀ HƯỚNG PHÁT TRIỂN	124
6.1. Kết quả đạt được của dự án	124
6.2. Những khó khăn, vướng mắc trong quá trình thực hiện	127
1. Bài toán rò rỉ bộ nhớ (Memory Leak) và nghẽn luồng xử lý đồ họa (EDT) trên Java Swing	128
2. Thách thức đồng bộ hóa và quản lý khóa mật mã đầu cuối (E2EE) đa thiết bị.....	128
3. Khó khăn trong việc định cấu hình định tuyến luồng mạng WebRTC qua hạ tầng NAT của Jitsi Videobridge (JVB)	129
6.3. Hạn chế của hệ thống hiện tại	130
6.4. Hướng phát triển và mở rộng trong tương lai.....	130
1. Tối ưu hóa Hiệu năng Local AI: Tăng tốc phân cứng đa nền tảng và Cơ chế RAG phi tập trung.....	130
2. Tối ưu hạ tầng: Kiến trúc Microservices kết hợp Kubernetes (K8s).....	131
3. Mô hình luồng dữ liệu kiến trúc tương lai	132
TÀI LIỆU THAM KHẢO.....	133

Lời cảm ơn

Lời đầu tiên, em xin được bày tỏ lòng biết ơn sâu sắc nhất đến Ban Giám hiệu Trường, Ban Chủ nhiệm Khoa Công nghệ thông tin, cùng toàn thể quý Thầy, Cô giáo đã tận tình truyền đạt kiến thức, nền tảng tư duy khoa học và tạo mọi điều kiện thuận lợi nhất cho em trong suốt quá trình học tập và nghiên cứu tại trường. Những tri thức quý báu đó là hành trang vững chắc để em có thể hoàn thành báo cáo đồ án tốt nghiệp với đề tài: **“Secure Chat - Hệ thống nhắn tin bảo mật tích hợp AI & Video Call”**.

Đặc biệt, em xin gửi lời cảm ơn chân thành và sâu sắc nhất tới **giáo viên hướng dẫn Nguyễn Tuấn Anh**, người đã trực tiếp hướng dẫn, định hướng đề tài và tận tình chỉ bảo em trong suốt quá trình thực hiện đồ án. Sự nghiêm túc trong nghiên cứu khoa học, những góp ý chuyên môn quý báu của thầy/cô về kiến trúc hệ thống Client-Server, giải pháp mã hóa đầu cuối (E2EE), cũng như phương án tối ưu hóa hạ tầng Docker (Self-hosted) là kim chỉ nam giúp em vượt qua các thách thức kỹ thuật phức tạp và hoàn thành đồ án đúng tiến độ.

Em cũng xin gửi lời cảm ơn đến các kỹ sư và chuyên gia công nghệ tại các cộng đồng nguồn mở (Jitsi Meet, Ollama, FlatLaf). Những tài liệu kỹ thuật và giải pháp của họ đã giúp em giải quyết được bài toán tích hợp hạ tầng hội nghị truyền hình (Prosody, Jicofo, JVB) cũng như tối ưu hóa hiệu năng của mô hình ngôn ngữ lớn (Llama 3 / Mistral) chạy cục bộ trên môi trường giới hạn tài nguyên của Client.

Cuối cùng, em xin cảm ơn gia đình, bạn bè và các bạn học cùng khóa đã luôn đồng hành, chia sẻ khó khăn và đóng góp những ý kiến trải nghiệm người dùng quý báu để hệ thống được hoàn thiện cả về mặt công nghệ lẫn giao diện UI/UX.

Mặc dù đã dành nhiều thời gian, tâm huyết và cố gắng nỗ lực tối đa để hoàn thành đồ án một cách hoàn chỉnh nhất; song do kiến thức bản thân và thời gian nghiên cứu còn những giới hạn nhất định, đồ án chắc chắn không tránh khỏi những thiếu sót. Em rất mong nhận được những ý kiến đóng góp, chỉ bảo quý báu của quý Thầy, Cô trong Hội đồng chấm đồ án tốt nghiệp để hệ thống Secure **Chat** được hoàn thiện và có thể ứng dụng vào thực tiễn một cách hiệu quả hơn.

Em xin chân thành cảm ơn!

Lời mở đầu

Trong kỷ nguyên chuyển đổi số và sự bùng nổ của các giải pháp làm việc từ xa, nhu cầu về một hệ thống liên lạc nội bộ đảm bảo tính an toàn dữ liệu, tối ưu hóa hiệu năng và tích hợp trí tuệ nhân tạo là vô cùng cấp thiết. Đồ án nghiên cứu và phát triển hệ thống **“SECURE CHAT - Hệ thống nhắn tin bảo mật tích hợp AI & Video Call”** nhằm giải quyết triệt để các thách thức về rò rỉ dữ liệu, phụ thuộc vào hạ tầng cloud bên thứ ba và tối ưu hóa trải nghiệm người dùng trong môi trường doanh nghiệp. Hệ thống được triển khai dựa trên kiến trúc Client-Server, kết hợp hạ tầng Container hóa (Docker Self-hosted) để đảm bảo khả năng mở rộng và toàn quyền kiểm soát tài nguyên.

Về mặt kỹ thuật, cốt lõi của đồ án tập trung vào việc hiện thực hóa năm trụ cột công nghệ chính:

Cơ chế mã hóa đầu cuối (E2EE): Đồ án triển khai một hệ thống mật mã lai (Hybrid Cryptosystem) tối ưu. Dữ liệu và nội dung hội thoại được bảo mật tuyệt đối bằng thuật toán AES-256 trong chế độ GCM, đảm bảo song song tính bí mật và toàn vẹn dữ liệu. Khóa phiên (Session Key) của AES được luân chuyển an toàn giữa các Client nhờ kỹ thuật mã hóa công khai RSA-2048. Điểm mấu chốt trong kiến trúc này là khóa bí mật (Private Key) của người dùng được lưu trữ hoàn toàn cục bộ; hệ thống sử dụng hàm PBKDF2 để dẫn xuất khóa từ mật khẩu, thực hiện mã hóa khóa bí mật ngay trên thiết bị trước khi lưu trữ, cam kết không gửi khóa về phía Server.

Hạ tầng Video/Audio Conference: Hệ thống tích hợp trực tiếp giải pháp Jitsi Meet Stack trên hạ tầng tự vận hành (Self-hosted) thay vì lệ thuộc vào các API thương mại. Cấu trúc cuộc gọi được điều phối bởi ba thành phần: Prosody (quản lý báo hiệu XMPP), Jicofo (phân bổ phòng họp) và Jitsi Videobridge (JVB). Trong đó, JVB đóng vai trò là một SFU (Selective Forwarding Unit), thực hiện chuyển tiếp luồng media mà không cần giải mã hay trộn luồng, giúp tối ưu hóa tài nguyên máy chủ và giảm thiểu độ trễ so với mô hình MCU truyền thống.

Trợ lý ảo AI cục bộ (Local AI): Để ngăn chặn nguy cơ rò rỉ thông tin từ các mô hình ngôn ngữ lớn (LLM) đám mây, đồ án sử dụng framework Ollama chạy trực tiếp trên hạ tầng nội bộ. Hệ thống vận hành các mô hình như Llama 3 hoặc Mistral đã được lượng tử hóa (4-bit/8-bit) để tương thích với phần cứng. Client giao tiếp với AI qua kênh API bất đồng bộ riêng biệt, đảm bảo mọi dữ liệu nhạy cảm của doanh nghiệp luôn nằm trong phạm vi kiểm soát nội bộ.

Giao thức truyền file an toàn: Quy trình gửi nhận tệp tin được bảo mật bằng cơ chế sinh khóa động. Khi có yêu cầu, hệ thống tự động khởi tạo khóa dùng một lần FileTransferKey (dựa trên cơ chế Salt và SHA-256). Dữ liệu tệp được chia nhỏ thành các đoạn (chunks) và mã hóa AES-256 trước khi truyền qua Socket. Máy chủ chỉ đóng vai trò lưu trữ tạm thời và điều phối các khối dữ liệu đã mã hóa; chỉ Client sở hữu FileTransferKey hợp lệ (nhận qua kênh chat E2EE) mới có thể giải mã và khôi phục tệp tin gốc.

TỪ ĐIỂN THUẬT NGỮ (GLOSSARY)

Dưới đây là danh mục các thuật ngữ kỹ thuật, giao thức, thuật toán và thành phần kiến trúc được sử dụng trong hệ thống Secure Chat, cùng với định nghĩa và phân tích luồng vận hành chi tiết dưới góc độ kỹ thuật.

1. Các thuật ngữ về An toàn thông tin và Mã hóa

E2EE (End-to-End Encryption - Mã hóa đầu cuối): Cơ chế bảo mật đảm bảo dữ liệu (tin nhắn, tệp tin) được mã hóa ngay tại thiết bị nguồn (Client gửi) và chỉ được giải mã tại thiết bị đích (Client nhận). Trong hệ thống Secure Chat, máy chủ trung gian (Server MySQL/Java) chỉ đóng vai trò lưu trữ và định tuyến các gói tin đã mã hóa dưới dạng bản mã (Ciphertext), hoàn toàn không thể đọc được nội dung gốc (Plaintext).

AES-256 (Advanced Encryption Standard - 256 bits): Thuật toán mã hóa đối xứng áp dụng kích thước khóa (Key size) là 256 bits. Hệ thống sử dụng AES-256 ở chế độ GCM (Galois/Counter Mode) để vừa đảm bảo tính bí mật (Confidentiality), vừa đảm bảo tính toàn vẹn (Integrity) của dữ liệu tin nhắn và tệp tin thông qua cơ chế mã hóa xác thực (AEAD).

RSA-2048 (Rivest–Shamir–Adleman - 2048 bits): Thuật toán mã hóa bất đối xứng sử dụng cặp khóa: Khóa công khai (Public Key) và Khóa bí mật (Private Key). Hệ thống sử dụng cặp khóa RSA-2048 cho pha bắt tay (Handshake) để trao đổi khóa đối xứng (Session Key của AES) một cách an toàn qua kênh truyền không bảo mật.

FileTransferKey (Khóa chuyển tệp tin): Một chuỗi byte ngẫu nhiên (chứa entropy cao) được hệ thống sinh tự động cho mỗi phiên tải tệp lên (Upload). Khóa này đóng vai trò là khóa đối xứng AES dùng để mã hóa tệp tin đó tại Client trước khi đẩy lên bộ lưu trữ tập trung của hạ tầng Docker. Khóa này chỉ được chia sẻ cho người nhận thông qua kênh tin nhắn đã mã hóa E2EE.

Key Derivation Function (KDF - Hàm dẫn xuất khóa): Hệ thống áp dụng thuật toán PBKDF2 (Password-Based Key Derivation Function 2) kết hợp với Salt (muối ngẫu nhiên) để băm mật khẩu đăng nhập của người dùng. Kết quả thu được vừa dùng để xác thực tài khoản, vừa làm khóa nền tảng để mã hóa nội bộ Private Key của cấu trúc RSA trước khi lưu xuống bộ nhớ cục bộ của Client.

2. Hạ tầng và Giao thức truyền thông đa phương tiện

Jitsi Meet Stack: Tập hợp các thành phần mã nguồn mở phân tán chịu trách nhiệm điều phối và thiết lập các phiên hội nghị truyền hình (Video/Audio Call). Hạ tầng bao gồm ba thực thể cốt lõi chạy trên các container Docker độc lập: Prosody, Jicofo, và JVB.

Prosody: Máy chủ XMPP (Extensible Messaging and Presence Protocol) trọng lượng nhẹ, đóng vai trò là xương sống báo hiệu (Signaling Server) cho hệ thống Jitsi, quản lý trạng thái hiện diện và kết nối của các Client tham gia cuộc gọi.

Jicofo (Jitsi Conference Focus): Thành phần chịu trách nhiệm quản lý logic thực thể phiên (Session focus). Nó phân bổ tài nguyên mạng, quản lý việc người dùng tham gia/rời phòng họp và điều phối các luồng media giữa Client với JVB.

JVB (Jitsi Videobridge): Thành phần hạ tầng định tuyến video (Selective Forwarding Unit - SFU). Thay vì trộn luồng video (MCU), JVB nhận luồng video từ mỗi Client và phân phối trực tiếp luồng đó đến các Client khác, giúp giảm thiểu đáng kể tải nguyên CPU tiêu thụ trên máy chủ.

3. Trí tuệ nhân tạo cục bộ (Local AI)

Ollama Framework: Kiến trúc nền tảng gọn nhẹ giúp đóng gói, quản lý và vận hành các mô hình ngôn ngữ lớn (LLM) cục bộ ngay trên hạ tầng Self-hosted của hệ thống. Ollama cung cấp hệ thống RESTful API nội bộ để ứng dụng Java Client tương tác trực tiếp mà không cần cấu hình phức tạp.

Llama 3 / Mistral: Các kiến trúc mô hình ngôn ngữ lớn mã nguồn mở có kích thước tham số tối ưu (7B/8B parameters), được tinh chỉnh (Fine-tuned) để chạy mượt mà trên môi trường máy chủ cục bộ hoặc thiết bị biên, đảm bảo xử lý ngôn ngữ tự nhiên tốt nhưng vẫn giữ tính bảo mật tuyệt đối cho dữ liệu hội thoại (Zero Cloud Data Leakage).

4. Kiến trúc và Giao diện ứng dụng

Java Swing: Bộ thư viện giao diện đồ họa (GUI) nền tảng của Java, được sử dụng để xây dựng ứng dụng Client. Hệ thống áp dụng mô hình kiến trúc MVC (Model-View-Controller) trên Swing để tách biệt logic xử lý mã hóa, kết nối mạng và hiển thị.

FlatLaf (Flat Look and Feel): Thư viện giao diện hiện đại cho Java Swing. Nó cung cấp các công cụ tối ưu hóa phần cứng để render các hiệu ứng đồ họa nâng cao như Dark mode (Chế độ tối), Glassmorphism (Hiệu ứng kính mờ), và các bong bóng chat dạng hình học mềm (Gradient Chat Bubbles).

Docker Self-hosted: Giải pháp container hóa toàn bộ hạ tầng Server-side (gồm MySQL Server, Jitsi Meet Stack, Ollama AI API). Cơ chế này giúp cô lập môi trường chạy ứng dụng, tăng khả năng mở rộng (Scalability) và tối giản quy trình triển khai hệ thống (CI/CD).

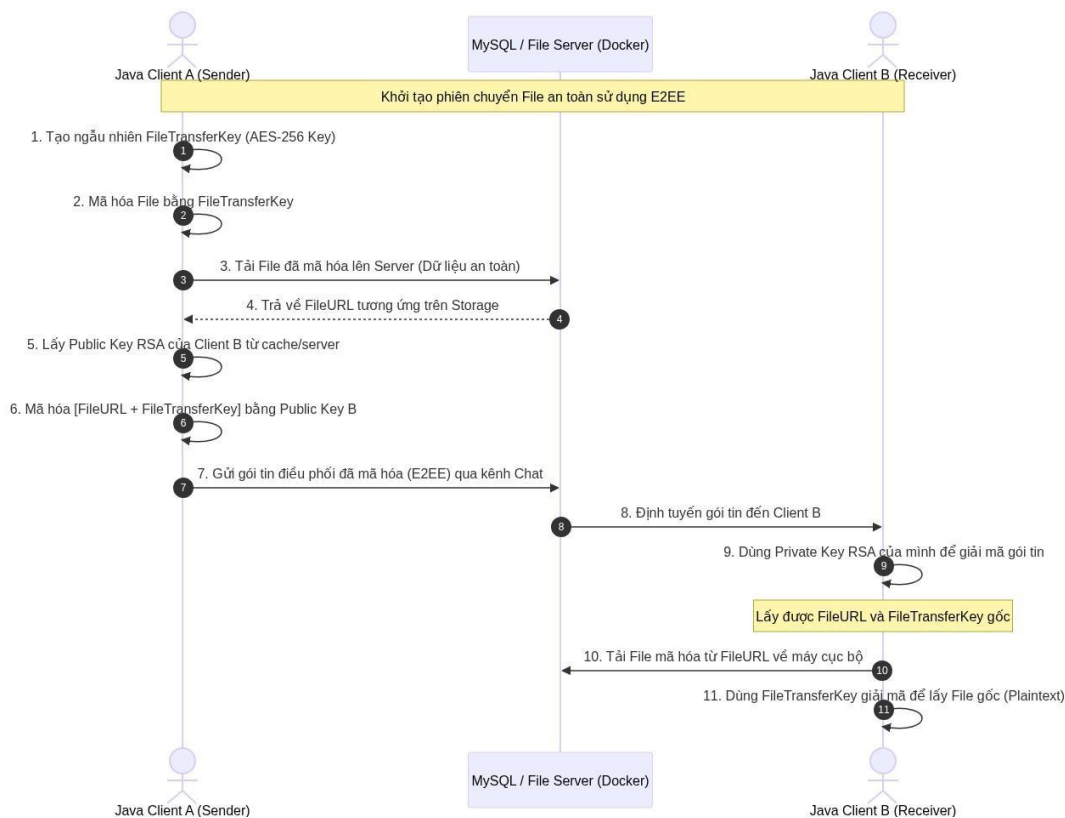
5. Bảng tổng hợp đối sánh và Luồng kiến trúc kỹ thuật

Để làm rõ cơ chế phối hợp hoạt động giữa các thành phần thuật ngữ nêu trên, dưới đây là bảng phân định vai trò kiến trúc và sơ đồ mô tả tuần tự luồng dữ liệu bảo mật (E2EE) kết hợp chuyển file an toàn trong hệ thống.

Phân định vai trò và phạm vi xử lý của các thuật ngữ cốt lõi

Nhóm thành phần	Thuật ngữ / Công nghệ	Môi trường thực thi	Vai trò cốt lõi trong hệ thống
An toàn dữ liệu	AES-256 (GCM Mode)	Client Side	Mã hóa bất đối xứng dữ liệu tin nhắn/file

Nhóm thành phần	Thuật ngữ / Công nghệ	Môi trường thực thi	Vai trò cốt lõi trong hệ thống
	RSA-2048	Client Side	Mã hóa và trao đổi Session Key đối xứng
Xử lý truyền thông	PBKDF2	Client Side	Bảo vệ Private Key lưu cục bộ
	Jitsi (Prosody, Jicofo)	Docker Container	Quản lý phiên giao tiếp và báo hiệu cuộc gọi
	Jitsi Videobridge (JVB)	Docker Container	Định tuyến luồng đa phương tiện (SFU)
Trí tuệ nhân tạo	Framework Ollama	Local Server Container	Quản lý và cung cấp REST API cho LLM
	Llama 3 / Mistral	Local Server Container	Xử lý suy luận ngữ cảnh hội thoại bảo mật
Giao diện người dùng	FlatLaf (Swing)	Client Machine	Kết xuất đồ họa nâng cao và tối ưu UI/UX



Hình 2 Sơ đồ tuần tự luồng chuyển đổi dữ liệu và áp dụng thuật toán mã hóa khi chuyển file an toàn.

Chương 1 Giới thiệu đề tài và Bối cảnh

1.1 Đặt vấn đề và Bối cảnh thực tiễn

Trong kỷ nguyên số hóa và sự bùng nổ của cuộc Cách mạng Công nghiệp 4.0, nhu cầu trao đổi thông tin trực tuyến đã trở thành một phần thiết yếu trong hoạt động vận hành của các tổ chức, doanh nghiệp cũng như đời sống hàng ngày của cá nhân. Tuy nhiên, sự tiện lợi này luôn song hành với các thách thức lớn về an toàn thông tin. Theo các báo cáo an ninh mạng toàn cầu, các vụ rò rỉ dữ liệu, tấn công nghe lén (Man-in-the-Middle - MitM) và khai thác thông tin người dùng trên các nền tảng đám mây (Cloud-based) ngày càng gia tăng với quy mô và thủ đoạn tinh vi.

Phần lớn các ứng dụng nhắn tin phổ biến hiện nay (như Messenger, Telegram, Zalo) dù đã tích hợp các biện pháp bảo mật nhất định, nhưng mô hình lưu trữ trung tâm trên hệ thống Cloud của nhà cung cấp dịch vụ vẫn tiềm ẩn nguy cơ bị khai thác dữ liệu từ bên trong (Insider Threats) hoặc bị can thiệp bởi bên thứ ba. Đặc biệt, đối với các tổ chức chính phủ, tài chính, hoặc các doanh nghiệp sở hữu bí mật kinh doanh, việc gửi các thông tin nhạy cảm, mã nguồn, hay tài liệu chiến lược lên các máy chủ công cộng là một rủi ro không thể chấp nhận.

Song song với đó, sự phát triển vượt bậc của Trí tuệ nhân tạo (AI) mà đặc biệt là các Mô hình ngôn ngữ lớn (LLM) đã mở ra xu hướng tích hợp trợ lý ảo vào hệ thống giao tiếp để tối ưu hóa hiệu suất làm việc (tóm tắt văn bản, dịch thuật, gợi ý phản hồi). Mặc dù vậy, việc gửi dữ liệu hội thoại lên các API đám mây thương mại (như OpenAI ChatGPT, Google Gemini) vô hình trung đã phá vỡ nguyên tắc bảo mật thông tin nội bộ.

Từ thực tiễn đó, yêu cầu đặt ra là phải xây dựng một hệ thống nhắn tin, gọi video nội bộ có khả năng tự chủ hoàn toàn về hạ tầng (Self-hosted), bảo mật tuyệt đối dữ liệu truyền tải thông qua mã hóa đầu cuối (E2EE), đồng thời tích hợp các mô hình AI vận hành cục bộ (Local AI) nhằm đáp ứng toàn diện cả hai bài toán: **Hiệu năng cộng tác** và **An toàn thông tin tối đa**. Dự án **“SECURE CHAT - Hệ thống nhắn tin bảo mật tích hợp AI & Video Call”** được nghiên cứu và phát triển nhằm giải quyết triệt để các thách thức nêu trên.

Phân tích giải pháp công nghệ và Kiến trúc hệ thống

Để hiện thực hóa mục tiêu bảo mật và tối ưu năng lực xử lý, hệ thống được thiết kế dựa trên kiến trúc Client-Server kết hợp hạ tầng container hóa Docker, phân tách rõ ràng giữa tầng xử lý nghiệp vụ, tầng truyền thông thời gian thực và tầng xử lý AI độc lập.

Cơ chế mã hóa đầu cuối (End-to-End Encryption - E2EE): Hệ thống áp dụng mô hình mã hóa lai (Hybrid Encryption) kết hợp giữa thuật toán đối xứng và bất đối xứng nhằm tối ưu hóa giữa tốc độ xử lý và mức độ an toàn:

Mã hóa dữ liệu (AES-256): Toàn bộ nội dung tin nhắn và tệp tin truyền tải được mã hóa cục bộ tại phía Client bằng thuật toán AES (Advanced Encryption Standard) với độ dài khóa 256-bit ở chế độ GCM (Galois/Counter Mode) để đảm bảo cả tính bảo mật (Confidentiality) lẫn tính toàn vẹn (Integrity) của dữ liệu.

Trao đổi khóa (RSA-2048): Để phân phối khóa đối xứng (Session Key/AES Key) giữa các Client một cách an toàn mà không bị lộ trên đường truyền, hệ thống sử dụng thuật toán mã hóa bất đối xứng RSA với độ dài khóa 2048-bit. Cặp khóa Công khai (Public Key) và Khóa bí mật (Private Key) được tạo lập ngay khi người dùng đăng ký tài khoản.

Quản lý Khóa nội bộ (Local Key Management): Khóa Public Key được đẩy lên Server để phục vụ quá trình định tuyến mã hóa, trong khi Khóa Private Key được lưu trữ nghiêm ngặt tại Client, được mã hóa bằng một mã băm (Hash/PBKDF2) từ mật khẩu của người dùng trước khi ghi vào bộ nhớ cục bộ, đảm bảo ngay cả quản trị viên hệ thống (Server Administrator) cũng không thể đọc được nội dung hội thoại.

Hạ tầng truyền thông đa phương tiện (Video/Audio Call): Hội nghị truyền hình được xây dựng dựa trên giao thức WebRTC thông qua việc tích hợp giải pháp mã nguồn mở **Jitsi Meet Stack**, phân rã thành các thành phần container hóa chạy trên Docker bao gồm:

Prosody: XMPP Server chịu trách nhiệm quản lý phiên (Session Signaling), xác thực người dùng và thiết lập trạng thái hiện diện.

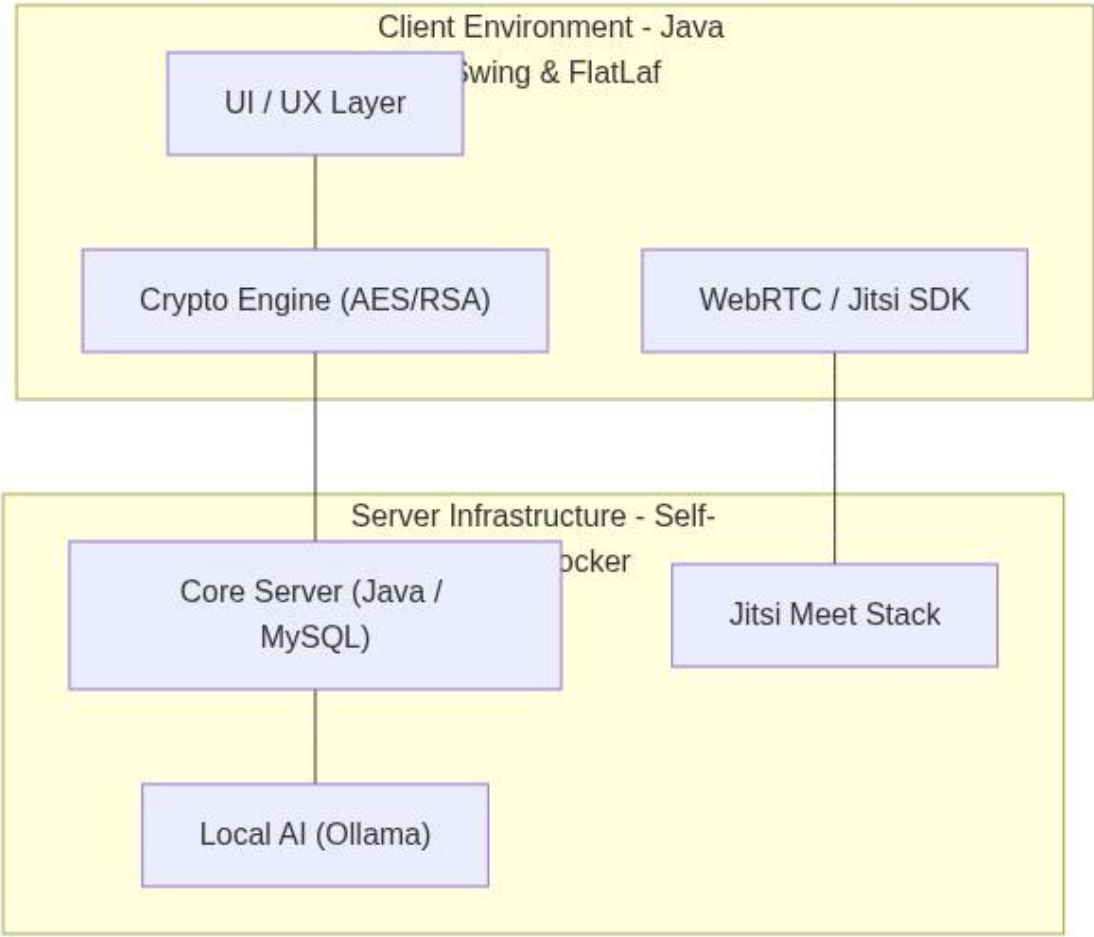
Jicofo (Jitsi Conference Focus): Thành phần điều phối và quản lý các phòng họp, phân bổ tài nguyên luồng media.

JVB (Jitsi Videobridge): Bộ chuyển mạch video (Selective Forwarding Unit - SFU). Thay vì trộn các luồng video lại với nhau gây tốn tài nguyên Server (như kiến trúc MCU), JVB chỉ nhận luồng media từ các Client và chuyển tiếp trực tiếp đến các Client khác trong phòng, tối ưu hóa băng thông và giảm thiểu độ trễ (Latency) xuống mức <100ms trong điều kiện mạng tiêu chuẩn.

Tích hợp Trí tuệ nhân tạo cục bộ (Local AI Ecosystem): Nhằm loại bỏ hoàn toàn nguy cơ rò rỉ dữ liệu hội thoại ra Internet, hệ thống tích hợp framework **Ollama** để chạy các mô hình ngôn ngữ lớn nguồn mở như Llama 3 (8B) hoặc Mistral (7B) ngay trên hạ tầng Server của tổ chức. Hệ thống Client gửi truy vấn (Prompt) thông qua một cổng API bảo mật nội bộ được thiết lập riêng. Toàn bộ vòng đời của dữ liệu (Data Lifecycle) từ lúc tiếp nhận, xử lý tokenization, suy luận (Inference) cho đến khi trả về kết quả đều diễn ra cục bộ trong mạng nội bộ, đáp ứng tiêu chuẩn khắt khe về quyền riêng tư.

Cơ chế chuyển tệp an toàn (Secure File Transfer): Khi người dùng chia sẻ một tệp tin, Client sẽ tự động sinh ngẫu nhiên một khóa dùng một lần có tên **FileTransferKey**. Tệp tin được mã hóa AES-256 bằng khóa này trước khi tải lên Server lưu trữ dưới dạng các khối nhị phân mã hóa (Encrypted Blobs). Khóa **FileTransferKey** sau đó được mã hóa bằng khóa RSA Public Key của người nhận và gửi qua kênh chat E2EE. Người nhận sau khi tải khối nhị phân về sẽ dùng Private Key của mình để giải mã lấy **FileTransferKey**, từ đó khôi phục lại tệp tin nguyên bản.

Kiến trúc tổng thể và Luồng dữ liệu kỹ thuật: Mối quan hệ tương tác và luồng vận hành giữa các thành phần trong hệ thống được mô tả chi tiết thông qua sơ đồ kiến trúc dưới đây:



Hình 1.1: Sơ đồ kiến trúc phân tầng hệ thống Secure Chat

Để làm rõ hơn các công nghệ được áp dụng và vai trò chiến lược của chúng trong việc giải quyết các bài toán kỹ thuật của đề tài, bảng dưới đây tổng hợp các thành phần cốt lõi của hệ thống:

Bảng 1.1: Tổng hợp các công nghệ cốt lõi và vai trò trong hệ thống

Thành phần	Công nghệ sử dụng	Vai trò kỹ thuật trong hệ thống
Ứng dụng Client	Java Swing, Thư viện FlatLaf	Cung cấp giao diện đồ họa hiện đại (Dark mode, hiệu ứng Glass Morphism), xử lý logic nghiệp vụ tại máy trạm và trực tiếp thực thi các tác vụ mã hóa/giải mã để đảm bảo nguyên tắc E2EE.
Cơ sở dữ liệu	MySQL	Lưu trữ thông tin tài khoản, danh bạ, các khóa công khai (Public Key) và lịch sử tin nhắn đã mã hóa dưới dạng chuỗi Base64.
Hạ tầng Container	Docker, Docker Compose	Đóng gói độc lập các dịch vụ (Core Server, Jitsi, Ollama), giúp hệ thống dễ dàng triển khai (Self-hosted) và mở rộng quy mô một cách linh hoạt.
Truyền thông Gọi Video	Jitsi Meet Stack (Prosody, Jicofo, JVB)	Định tuyến luồng đa phương tiện (Audio/Video) thời gian thực theo kiến trúc SFU, giảm tải cho Server và đảm bảo chất lượng cuộc gọi độ phân giải cao.
Xử lý AI	Framework Ollama (Llama 3 / Mistral)	Thực hiện suy luận ngôn ngữ tự nhiên ngay tại máy chủ cục bộ, không kết nối Internet, bảo vệ tuyệt đối dữ liệu doanh nghiệp.

Mục tiêu và Đóng góp của đề tài

Mục tiêu tổng quát của đề tài là xây dựng hoàn chỉnh một giải pháp phần mềm nhắn tin và hội họp trực tuyến có tính thực tiễn cao, kết hợp hài hòa giữa các tiêu chuẩn an ninh nghiêm ngặt và trải nghiệm người dùng tối ưu. Cụ thể, đề tài hướng tới các mục tiêu kỹ thuật sau:

Thiết kế và cài đặt thành công giao thức mã hóa đầu cuối (E2EE) hoạt động ổn định trên nền tảng ứng dụng desktop Java Swing, đảm bảo tốc độ mã hóa/giải mã tin nhắn thời gian thực mà không gây treo hoặc trễ giao diện (UI Thread).

Triển khai và tối ưu hóa hạ tầng Jitsi Meet Stack trên môi trường Docker, cấu hình chính xác các tham số mạng và NAT để cho phép thiết lập cuộc gọi Video/Audio mượt mà giữa các Client nằm sau các lớp tường lửa khác nhau.

Tích hợp thành công mô hình Local LLM thông qua Ollama, tối ưu hóa cấu hình phần cứng (RAM/VRAM) tại Server để đạt tốc độ xử lý sinh văn bản (Token Generation Rate) phù hợp cho việc tương tác thời gian thực của trợ lý AI.

Xây dựng giao diện người dùng trực quan, thẩm mỹ bằng cách tùy biến sâu thư viện FlatLaf, phá bỏ giới hạn thô cứng truyền thống của Java Swing để đạt tới trải nghiệm hiện đại tương đương các ứng dụng Electron hay Native hiện nay.

Kết quả của đề tài đóng vai trò như một mô hình tham chiếu thực nghiệm có giá trị cho các nghiên cứu và dự án phát triển ứng dụng bảo mật nội bộ, chứng minh khả năng tự chủ công nghệ từ khâu hạ tầng, truyền thông cho đến trí tuệ nhân tạo trong bối cảnh an toàn thông tin ngày càng trở thành yếu tố sống còn của mọi tổ chức.

1.2. Nhu cầu tích hợp AI (Trợ lý ảo) và giao tiếp đa phương tiện an toàn

Sự phát triển mạnh mẽ của kỷ nguyên số đặt ra những thách thức mới đối với các hệ thống thông tin liên lạc. Người dùng hiện nay không chỉ đòi hỏi một kênh truyền tin nhắn thuần túy mà còn yêu cầu khả năng tối ưu hóa hiệu suất làm việc thông qua trí tuệ nhân tạo (AI) và các phương thức giao tiếp đa phương tiện thời gian thực (Real-time Multimedia Communication). Tuy nhiên, việc tích hợp các công nghệ tiên tiến này vào một môi trường bảo mật cao đòi hỏi những giải pháp kiến trúc chuyên sâu để giải quyết mâu thuẫn giữa tính tiện ích và quyền riêng tư dữ liệu.

1. Nguy cơ an toàn thông tin khi tích hợp Trợ lý ảo AI dựa trên Điện toán đám mây

Trong các kiến trúc ứng dụng thông thường, việc tích hợp các mô hình ngôn ngữ lớn (LLM) như GPT-4 hay Claude thường được thực hiện qua các API thương mại dựa trên nền tảng Cloud (Cloud-based AI). Mô hình này tồn tại những lỗ hổng bảo mật nghiêm trọng đối với các tổ chức yêu cầu tính bảo mật dữ liệu tuyệt đối:

Sự rò rỉ dữ liệu qua kênh thứ ba (Data Leakage): Để AI có thể xử lý và đưa ra phản hồi chính xác, toàn bộ nội dung hội thoại, dữ liệu doanh nghiệp hoặc thông tin cá nhân của người dùng phải được chuyển sang dạng văn bản rõ (Plaintext) và gửi lên máy chủ của nhà cung cấp dịch vụ đám mây. Tiến trình này phá vỡ hoàn toàn nguyên lý cốt lõi của mã hóa đầu cuối (E2EE) vì luồng dữ liệu bị giải mã tại một nút trung gian nằm ngoài tầm kiểm soát của tổ chức.

Nguy cơ từ việc tái huấn luyện mô hình (Model Retraining Risks): Nhiều điều khoản dịch vụ đám mây cho phép nhà cung cấp sử dụng dữ liệu từ các yêu cầu API để tiếp tục huấn luyện, tinh

chính mô hình. Điều này dẫn đến nguy cơ thông tin chiến lược hoặc bí mật kinh doanh bị lưu vết vô định hình và vô tình hiển thị trong kết quả trả về của những người dùng khác trên thế giới.

Do đó, nhu cầu tích hợp một hệ thống **Local AI (Trí tuệ nhân tạo cục bộ)** thông qua framework **Ollama** là một tất yếu kỹ thuật. Bằng việc vận hành trực tiếp các mô hình mã nguồn mở như Llama 3 hoặc Mistral trên hạ tầng Docker Self-hosted biệt lập, hệ thống thiết lập một ranh giới bảo mật tuyệt đối. Luồng dữ liệu Prompt từ Client truyền đến AI hoàn toàn chạy trong mạng nội bộ, loại bỏ hoàn toàn các vector tấn công khai thác dữ liệu từ xa.

2. Thách thức bảo mật trong truyền thông đa phương tiện thời gian thực (Video/Audio Call)

Khác với tin nhắn văn bản vốn có dung lượng nhỏ và dễ xử lý bằng các thuật toán mã hóa khối (Block ciphers), dữ liệu đa phương tiện (Video/Audio Streams) cấu thành từ các luồng bit liên tục, dung lượng lớn và yêu cầu khắt khe về thời gian thực. Việc thiết lập một kênh giao tiếp đa phương tiện an toàn đối mặt với hai bài toán kiến trúc lớn:

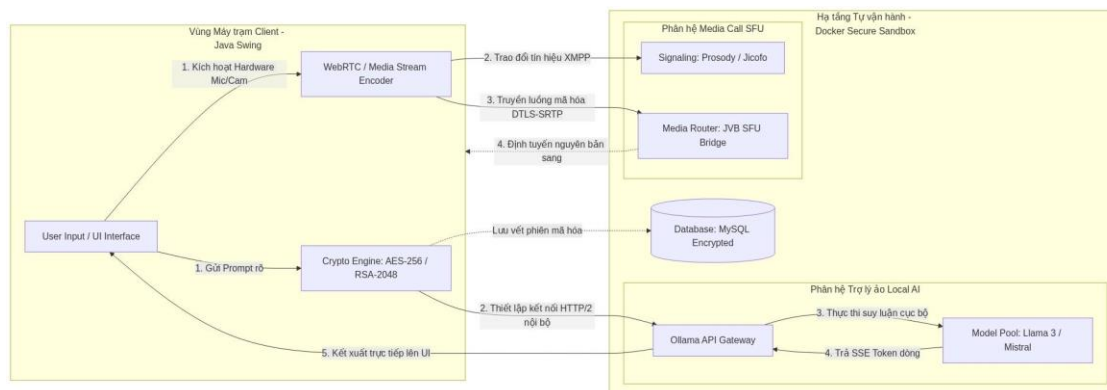
Sự đánh đổi giữa hiệu năng và tính bảo mật: Quá trình mã hóa và giải mã liên tục các khung hình (Frames) ở độ phân giải cao sử dụng các thuật toán mã hóa dòng có thể gây thắt nút cổ chai tài nguyên (Resource bottleneck) tại chip xử lý của máy trạm Client, dẫn đến hiện tượng trễ tín hiệu (Latency) hoặc mất đồng bộ âm thanh - hình ảnh (Audio-Video desynchronization).

Lỗ hổng từ kiến trúc chuyển tiếp dữ liệu trung gian: Nếu sử dụng kiến trúc máy chủ trộn luồng MCU (Multipoint Control Unit) truyền thống, máy chủ buộc phải giải mã toàn bộ dữ liệu stream từ các nhánh gửi, thực hiện trộn hình ảnh rồi mã hóa lại để gửi đi. Quá trình này biến máy chủ trung gian thành một "Điểm yếu duy nhất" (Single Point of Failure). Nếu máy chủ bị chiếm quyền kiểm soát, toàn bộ nội dung cuộc gọi sẽ bị lộ lọt.

Hệ thống Secure Chat giải quyết triệt để nhu cầu này bằng cách tích hợp nền tảng **Jitsi Meet Stack (Prosody, Jicofo, JVB)** hoạt động theo kiến trúc SFU (Selective Forwarding Unit). Luồng đa phương tiện từ Client được bảo vệ bằng giao thức **DTLS-SRTP** ngay tại tầng truyền tải, đảm bảo dữ liệu khi đi qua Jitsi Videobridge (JVB) chỉ là các gói tin đã mã hóa, máy chủ trung gian hoàn toàn không có khả năng đọc được nội dung gốc của phiên Call.

3. Sơ đồ Luồng dữ liệu và Ranh giới bảo mật hệ thống

Để làm rõ kiến trúc vận hành song song của phân hệ Local AI và phân hệ Video Call an toàn trong mối tương quan với ranh giới bảo mật hạ tầng Docker, luồng di chuyển của dữ liệu được đặc tả qua sơ đồ dưới đây:



Hình 4: Luồng kiến trúc dữ liệu và ranh giới cô lập bảo mật của phân hệ Local AI và Media Call.

4. Ma trận phân tích sự cần thiết kỹ thuật và Giải pháp đối sánh

Sự dịch chuyển từ các giải pháp tích hợp bên thứ ba (Third-party Cloud Services) sang giải pháp tự vận hành, cô lập dữ liệu (Self-hosted & Local Processing) trong hệ thống được luận chứng khoa học thông qua Bảng 1.

Ma trận đối sánh nhu cầu tích hợp công nghệ và giải pháp kiến trúc

Nhu cầu chức năng	Giải pháp thông thường (Cloud / Centralized)	Rủi ro an toàn thông tin phát sinh	Giải pháp kiến trúc Secure Chat (Self-hosted / Local)
Tích hợp Trợ lý ảo thông minh	Kết nối API mở rộng tới OpenAI GPT / Anthropic Claude.	Rò rỉ văn bản rõ hội thoại; vi phạm quyền riêng tư dữ liệu nghiêm trọng; phụ thuộc đường truyền Internet.	Vận hành Ollama API kết hợp mô hình Llama 3 / Mistral chạy hoàn toàn trên RAM/VRAM nội bộ của hạ tầng Docker.
Đàm thoại video đa người dùng	Sử dụng nền tảng trộn luồng MCU hoặc sử dụng API Zoom / Agora.	Máy chủ trung gian có quyền giải mã dữ liệu cuộc gọi; nguy cơ nghe lén tại các điểm nút điều phối Cloud.	Áp dụng Jitsi Stack với cơ chế định tuyến SFU (JVB). Luồng đa phương tiện mã hóa DTLS-SRTP xuyên suốt từ đầu cuối.
Lưu trữ và Truyền tệp tin	Tải trực tiếp lên các kho lưu trữ đám mây công cộng (AWS S3, Google Drive).	Quản trị viên nhà cung cấp Cloud có quyền tiếp cận file; nguy cơ quét mã độc tự	Mã hóa file bằng FileTransferKey đối xứng ngẫu nhiên tại máy trạm trước khi

Nhu cầu chức năng	Giải pháp thông thường (Cloud / Centralized)	Rủi ro an toàn thông tin phát sinh động làm lộ nội dung.	Giải pháp kiến trúc Secure Chat (Self-hosted / Local) đẩy lên bộ lưu trữ cục bộ của Docker.
-------------------	--	--	---

1.3 Phân tích nền tảng Telegram, Signal

Phân tích nền tảng Telegram, Signal

Nền tảng Telegram và Giao thức MTProto Telegram là một trong những ứng dụng nhắn tin có quy mô người dùng lớn nhất hiện nay, hoạt động dựa trên mô hình kiến trúc phân tán đa vùng lưu trữ đám mây (Cloud-based platform). Về mặt kỹ thuật, Telegram sử dụng một giao thức mã tự phát triển có tên gọi là **MTProto** (hiện tại là phiên bản MTProto 2.0).

Kiến trúc bảo mật của Telegram được chia làm hai chế độ hội thoại rõ rệt:

Hội thoại đám mây (Cloud Chats): Đây là chế độ mặc định của Telegram. Toàn bộ tin nhắn, tệp tin đa phương tiện được mã hóa giữa Client và Server thông qua giao thức MTProto, sau đó được lưu trữ tại hạ tầng đám mây của Telegram. Mặc dù dữ liệu trên đĩa cứng của máy chủ được mã hóa và các khóa giải mã được phân tán ở các thực thể pháp lý khác nhau nhằm chống can thiệp vật lý, nhưng về mặt kiến trúc lý thuyết, Telegram Server vẫn nắm giữ khả năng giải mã dữ liệu rõ (Plaintext) để phục vụ tính năng đồng bộ hóa đa thiết bị (Multi-device Synchronization) và tìm kiếm toàn văn (Full-text Search). Do đó, chế độ này không đạt tiêu chuẩn mã hóa đầu cuối (Non-E2EE).

Hội thoại bí mật (Secret Chats): Đây là chế độ áp dụng cơ chế mã hóa đầu cuối (E2EE). MTProto thiết lập kênh truyền bảo mật trực tiếp giữa hai thiết bị đầu cuối bằng giải thuật trao đổi khóa Diffie-Hellman (DH) kết hợp với hàm băm mật mã SHA-256. Dữ liệu sau đó được mã hóa đối xứng bằng thuật toán AES-256 ở chế độ IGE (Infinite Garble Extension). Bản dịch mã của Secret Chats chỉ lưu cục bộ trên thiết bị và không được đồng bộ hóa lên máy chủ đám mây.

Mặc dù có hiệu năng truyền tin tối ưu, kiến trúc của Telegram vẫn bộc lộ những hạn chế lớn dưới góc độ an ninh học thuật: Giao thức mã hóa đối xứng AES-IGE là một biến thể không tiêu chuẩn, thiếu các chứng nhận kiểm định an toàn quốc tế rộng rãi so với AES-GCM. Thêm vào đó, việc cấu hình mặc định không bật E2EE tạo ra nguy cơ rò rỉ thông tin lớn nếu tài khoản người dùng bị chiếm quyền kiểm soát phiên (Session Hijacking) trên máy chủ.

Nền tảng Signal và Giao thức Signal Protocol Ngược lại hoàn toàn với mô hình lưu trữ trung tâm của Telegram, Signal được xem là tiêu chuẩn vàng (Gold Standard) trong lĩnh vực truyền thông bảo mật nhờ việc áp dụng cơ chế mã hóa đầu cuối mặc định cho toàn bộ các loại hình dữ liệu (tin nhắn văn bản, cuộc gọi, tệp tin) thông qua giao thức mã hóa mã nguồn mở **Signal Protocol** (tiền thân là TextSecure Protocol).

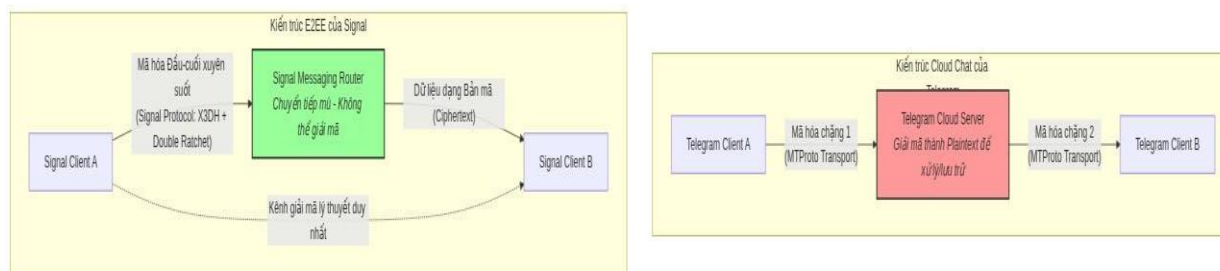
Cốt lõi toán học và kiến trúc của Signal Protocol dựa trên hai thực thể giải thuật cấp cao:

Giải thuật X3DH (Extended Triple Diffie-Hellman): Giải quyết bài toán thiết lập khóa chung bí mật giữa hai thực thể mà không yêu cầu cả hai phải cùng trực tuyến ('Asynchronous Key Exchange'). X3DH sử dụng tập hợp các khóa công khai ngắn hạn (Ephemeral Keys) và dài hạn (Identity Keys) dựa trên đường cong Elliptic Curve Diffie-Hellman (X25519), giúp chống lại các cuộc tấn công giả mạo danh tính ngay tại bước thiết lập phiên đầu tiên.

Giải thuật Double Ratchet Algorithm (Hàm băm bánh răng kép): Đây là cơ chế quản lý khóa mang tính đột phá của Signal. Khóa phiên liên tục được biến đổi ('Ratchet') sau mỗi lượt tin nhắn được gửi hoặc nhận. Quá trình này được thực hiện thông qua sự kết hợp giữa một KDF Chain (Key Derivation Function) dựa trên cấu trúc hình học cây mật mã (KDF-backed Symmetric Ratchet) và một chu kỳ biến đổi bất đối xứng Diffie-Hellman (DH Ratchet).

Nhờ sự kết hợp này, Signal Protocol đạt được hai đặc tính an ninh tối cao trong mật mã học: Tính bảo mật hoàn hảo cho tương lai (**Perfect Forward Secrecy - PFS**) — nếu khóa phiên hiện tại bị lộ, tin tức cũng không thể giải mã các tin nhắn trong quá khứ; và Tính bảo mật phục hồi sau tổn thất (**Post-Quantum/Break-in Recovery**) — nếu một khóa tại thời điểm t bị bẻ gãy, hệ thống sẽ tự động sinh ra một entropy mới ở thời điểm $t + 1$ để cô lập vùng tổn thất, ngăn chặn việc đọc lén các tin nhắn tương lai. Máy chủ của Signal đóng vai trò hoàn toàn là một nút chuyển tiếp mù (Blind Router), không lưu giữ bất kỳ siêu dữ liệu (Metadata) nhạy cảm nào liên quan đến danh bạ hay lịch sử kết nối của người dùng.

So sánh đối chiếu luồng kiến trúc kỹ thuật Để làm rõ sự khác biệt bản chất trong luồng định tuyến và xử lý dữ liệu giữa hai nền tảng, sơ đồ dưới đây mô tả cấu trúc ranh giới giải mã của Telegram (Cloud Chat) và Signal (Mặc định E2EE):



Hình 5: Sơ đồ so sánh ranh giới giải mã dữ liệu giữa cấu trúc Cloud-centric (Telegram) và E2EE (Signal)

Dựa trên các phân tích chuyên sâu về mật giải thuật và mô hình vận hành, bảng dưới đây tổng hợp các tiêu chí kỹ thuật cốt lõi để đối chiếu chi tiết giữa hai nền tảng, làm cơ sở thực tiễn cho việc đề xuất kiến trúc hệ thống Secure Chat:

Bảng 1.2: Ma trận so sánh đặc tính kỹ thuật giữa Telegram và Signal

	Nền tảng Telegram	Nền tảng Signal
Tiêu chí phân tích		
Giao thức Mật mã	MTPProto 2.0 (Tự phát triển).	Signal Protocol (Tiêu chuẩn mở ngành mã hóa).

Tiêu chí phân tích	Nền tảng Telegram	Nền tảng Signal	
Cơ chế E2EE	Không bật mặc định (Chỉ áp dụng khi khởi tạo thủ công phiên Secret Chat).	Kích hoạt mặc định 100% cho toàn bộ hội thoại, nhóm chat và media.	
Thuật toán đối xứng	AES-256-IGE (Infinite Garble Extension).	AES-256-GCM kết hợp HKDF (HMAC-based Key Derivation Function).	
Đặc tính PFS \	Break-in Recovery	Chỉ có PFS trên luồng Secret Chat thông qua tính toán lại khóa DH theo chu kỳ thời gian.	Đạt tối đa cả hai đặc tính trên từng cấu trúc tin nhắn nhờ thuật toán Double Ratchet.
Mô hình hạ tầng	Đám mây công cộng tập trung (Hosted on Telegram Cloud infrastructure).	Đám mây công cộng (Hosted on Signal Server), lưu trữ phi trạng thái.	
Quản lý siêu dữ liệu (Metadata)	Lưu trữ danh bạ, địa chỉ IP, lịch sử đăng nhập thiết bị tập trung tại Server.	Ẩn danh hóa hoàn toàn bằng công nghệ Sealed Sender, không lưu vết siêu dữ liệu.	

Từ bảng so sánh trên, có thể nhận thấy mặc dù Signal đạt điểm tuyệt đối về mặt lý thuyết an toàn mật mã học, nhưng cả hai nền tảng đều có một điểm chung là vận hành dựa trên kiến trúc hạ tầng phụ thuộc vào máy chủ của nhà cung cấp (Public Cloud-based). Đối với bài toán an ninh nội bộ của các tổ chức cơ mật, việc không thể tự chủ hạ tầng lưu trữ (Self-hosted) và không có khả năng tích hợp các công cụ trí tuệ nhân tạo (Local AI) xử lý tại chỗ chính là khoảng trống công nghệ lớn. Đây là động lực cốt lõi để hệ thống **Secure Chat** kế thừa ưu điểm mã hóa lai mạnh mẽ tương tự Signal, đồng thời đóng gói bao bọc trong giải pháp Containerization của Docker để vận hành hoàn toàn độc lập cục bộ.

Chương 2: Cơ sở lý thuyết và công nghệ sử dụng

2.1 Java & Java Swing (Phát triển giao diện Client với FlatLaf)

Java Swing được chọn làm nền tảng phát triển Client nhờ tính đa nền tảng (Write Once, Run Anywhere) và khả năng can thiệp sâu vào giao diện mức thấp.

FlatLaf (Flat Look and Feel): Đây là thư viện hiện đại giúp “lột xác” giao diện Swing cũ kỹ thành giao diện phẳng (Flat Design) giống như các ứng dụng web hiện đại. **Dark Mode:** Hỗ trợ sẵn các bộ màu tối chuyên nghiệp. **Hiệu ứng Glassmorphism:** Kết hợp với độ trong suốt để tạo cảm giác sang trọng. **Hiệu năng:** FlatLaf cực nhẹ, không gây tốn tài nguyên GPU như JavaFX.

2.2 Môi trường thực thi (JDK 17+, JRE)

Môi trường thực thi đóng vai trò nền tảng quyết định hiệu năng, độ ổn định và khả năng bảo mật của thành phần Client (Java Swing) trong hệ thống Secure Chat. Dự án lựa chọn phiên bản Java Development Kit từ 17 trở lên (JDK 17+), cụ thể là các bản phân phối LTS (Long-Term Support) như OpenJDK hoặc Oracle JDK, làm môi trường biên dịch và chạy ứng dụng thông qua Java Runtime Environment (JRE). Việc lựa chọn này dựa trên các phân tích kỹ thuật chuyên sâu về kiến trúc máy ảo JVM (Java Virtual Machine) và các cải tiến cốt lõi sau:

1. Cơ chế quản lý bộ nhớ và Bộ thu gom rác (Garbage Collection - ZGC & G1GC)

Đối với một ứng dụng nhắn tin thời gian thực tích hợp luồng xử lý đa phương tiện (Video/Audio Call qua Jitsi) và mã hóa liên tục như Secure Chat, độ trễ hệ thống (Latency) và hiện tượng “Stop-the-World” (STW) của Garbage Collector là những yếu tố phân tích sống còn.

G1GC (Garbage-First Garbage Collector): Được tối ưu hóa mặc định từ Java 9+, G1GC phân chia bộ nhớ Heap thành các vùng (Regions) kích thước bằng nhau từ 1MB đến 32MB. Hệ thống Secure Chat tận dụng cơ chế tăng tốc thu gom của G1GC trên JDK 17 nhờ thuật toán dự đoán đích (predictive targeting), tự động tính toán thời gian tạm dừng mục tiêu (MaxGCPauseMillis) xuống dưới 200ms, giúp giao diện Java Swing không bị đóng băng khi xử lý các mảng Byte dữ liệu lớn từ luồng mã hóa AES-256.

ZGC (Z Garbage Collector): Là một cải tiến đột phá chuyển sang trạng thái sẵn sàng sản xuất (Production-ready) từ JDK 15+. ZGC thực hiện tất cả các chu kỳ thu gom rác (Marking, Relocation, Remapping) một cách song song (Concurrent) với các luồng ứng dụng (Application threads) thông qua kỹ thuật rào cản tải (Load Barriers) và con trỏ màu (Colored Pointers). Trong các phiên kết nối hội nghị truyền hình tải cao, việc kích hoạt cấu hình ZGC bằng tham số -XX:+UseZGC đảm bảo thời gian dừng STW không vượt quá 10ms, bất kể kích thước Heap lên tới hàng Gigabyte do bộ đệm luồng video (Video stream buffering).

2. Các tính năng ngôn ngữ và Tối ưu hóa kiến trúc hạ tầng

JDK 17 cung cấp các mô hình dữ liệu mới giúp tối giản mã nguồn, giảm thiểu lỗi runtime và tối ưu hóa hiệu năng giao tiếp mạng:

Java Records (JEP 395): Được sử dụng để định nghĩa các cấu trúc dữ liệu bất biến (Immutable Data Carriers) trong luồng DTO (Data Transfer Object) giữa Client và Server (ví dụ: các gói tin mã hóa `ChatMessageDTO`, `FileChunkDTO`). Bản chất bất biến của Record ngăn chặn triệt để các lỗi can thiệp dữ liệu luồng (Race conditions) và giúp JVM tối ưu hóa vùng nhớ Stack thay vì cấp phát trên Heap thông qua cơ chế phân tích thoát (Escape Analysis).

Sealed Classes (JEP 409): Cung cấp khả năng kiểm soát chặt chẽ kiến trúc phân cấp kế thừa. Trong module bảo mật, lớp `SecureToken` được khai báo dưới dạng `sealed` và chỉ cho phép các lớp `AccessToken`, `FileTransferKey` kế thừa. Điều này ngăn chặn việc mã độc tiêm nhập (Code

injection) bằng cách tạo ra các lớp con giả mạo nhằm đánh cắp khóa mật mã.

Strong Encapsulation (JEP 403): JDK 17 thực thi việc đóng gói mạnh mẽ các thành phần nội bộ của JDK, mặc định vô hiệu hóa việc truy cập trái phép qua cơ chế Reflection đối với các hàm nội bộ ngoại trừ cấu hình tường minh. Cơ chế này trực tiếp nâng cao tính an toàn cho ứng dụng Secure Chat trước các cuộc tấn công khai thác lỗ hổng bộ nhớ trong thời gian thực.

3. Kiến trúc tiền xử lý mật mã học qua JCA/JCE trên JDK 17

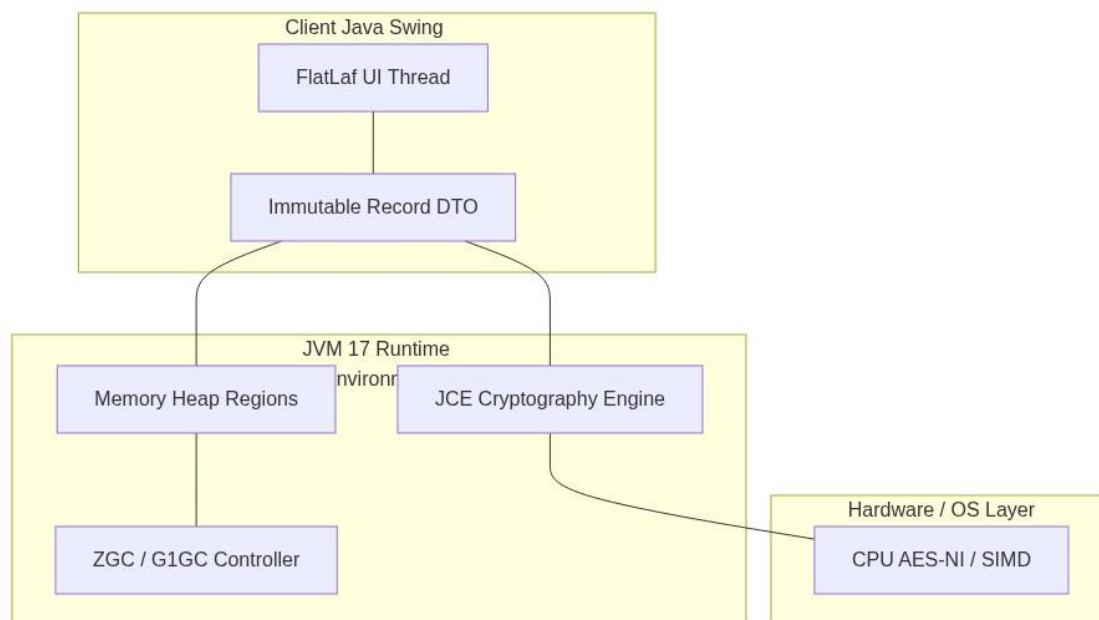
Một lý do kỹ thuật tiên quyết cho việc bắt buộc sử dụng JDK 17+ nằm ở kiến trúc mã hóa Java Cryptography Architecture (JCA) và Java Cryptography Extension (JCE). Hệ thống Secure Chat đòi hỏi thực thi liên tục các phép toán số học lớn cho RSA-2048 và các phép toán dịch bit trên mảng khối của AES-256 GCM.

Trong các phiên bản JDK cũ, việc tính toán mật mã phụ thuộc nặng nề vào các hàm thư viện phần mềm thuần túy, gây tiêu tốn tài nguyên CPU cực lớn. Từ JDK 17, JCA tích hợp sâu các tập lệnh nội tại (Intrinsic Functions) giúp JVM biên dịch trực tiếp mã bytecode thành mã máy tận dụng phần cứng chuyên dụng của CPU:

AES-NI (Advanced Encryption Standard New Instructions): JVM tự động ánh xạ các hàm mã hóa/giải mã AES trong JCE sang tập lệnh phần cứng AES-NI trên các dòng chip Modern x86_64 và ARM64 (Crypto Extensions), tăng tốc độ xử lý dữ liệu tệp tin gấp nhiều lần.

Vector API (Incubator - JEP 414): Hỗ trợ thực hiện các phép toán vector hóa song song SIMD (Single Instruction, Multiple Data), tối ưu hóa tốc độ tính toán mã xác thực thông điệp (MAC) và mã hóa dòng dữ liệu (Stream encryption).

Để mô tả cách thức môi trường thực thi (JRE/JVM 17) điều phối luồng dữ liệu, quản lý bộ nhớ Heap và tương tác với phần cứng khi nhận được một gói tin mã hóa từ socket, luồng kiến trúc kỹ thuật được thực hiện tuần tự theo sơ đồ dưới đây:



Hình 6: Kiến trúc tương tác phân tầng giữa Ứng dụng Client, Máy ảo JVM 17 và Lớp Phần cứng.

4. Bảng thông số cấu hình và Yêu cầu môi trường tối thiểu

Dựa trên các phân tích kiến trúc nêu trên, các thông số cấu hình tối ưu cho môi trường thực thi ứng dụng Client được đặc tả chi tiết trong Bảng 1.

Thông số kỹ thuật của môi trường thực thi JDK/JRE 17+ cho Client

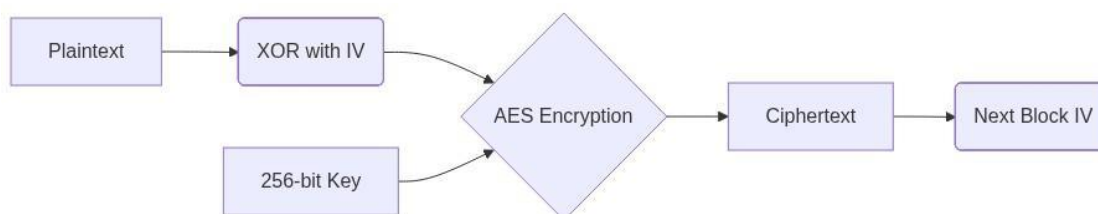
Thành phần cấu hình	Thông số / Phiên bản	Giải trình kỹ thuật mục tiêu
Java Runtime Version	OpenJDK 17.0.x LTS trở lên	Đảm bảo tính ổn định lâu dài, hỗ trợ đầy đủ các bản vá bảo mật cho các module mạng và mật mã.
Garbage Collector	-XX:+UseG1GC hoặc -XX:+UseZGC	Tối ưu hóa chu kỳ tạm dừng hệ thống (STW), triệt tiêu hiện tượng lag/giật khung hình khi render đồ họa FlatLaf.
Memory Allocation	-Xms512m -Xmx2g	Giới hạn vùng nhớ Heap ban đầu và tối đa, đáp ứng việc lưu trữ tạm thời các khối dữ liệu tệp tin khi giải mã.
Cryptographic Provider	SunJCE (Tích hợp sẵn trong JDK 17)	Kích hoạt cơ chế phần cứng AES-NI, tăng tốc xử lý giải mã luồng E2EE trực tiếp từ Socket.
GUI Rendering Engine	Pipeline - Dsun.java2d.d3d=true hoặc opengl=true	Tăng tốc phần cứng cho thư viện FlatLaf qua Direct3D hoặc OpenGL, hiển thị mượt mà hiệu ứng Glassmorphism.

2.3. AES-256 (Mã hóa đối xứng cho nội dung tin nhắn)

AES-256 là tiêu chuẩn mã hóa dữ liệu đối xứng được lựa chọn để bảo vệ nội dung tin nhắn trong hệ thống Secure Chat nhờ vào độ bảo mật cực cao và hiệu năng xử lý ấn tượng.

Đặc điểm kỹ thuật: **Độ dài khóa:** 256 bits (32 bytes), cung cấp 2^{256} tổ hợp khóa, miễn nhiễm với các cuộc tấn công vét cạn (Brute-force) trong tương lai gần. **Chế độ hoạt động:** Hệ thống sử dụng chế độ **Cipher Block Chaining (CBC)** kết hợp với PKCS5Padding để đảm bảo tính xáo trộn dữ liệu. **Vector khởi tạo (IV):** Mỗi tin nhắn được mã hóa với một IV ngẫu nhiên 16 bytes, đảm bảo cùng một nội dung tin nhắn sẽ có bản mã khác nhau mỗi lần gửi.

Tiêu chí	AES-128	AES-192	AES-256
Độ dài khóa	128 bit	192 bit	256 bit
Số vòng lặp	10	12	14
Độ an toàn	Cao	Rất cao	Tuyệt đối



Hình 7.: Luồng mã hóa AES trong chế độ CBC.

Kiến trúc Containerization với Docker & Docker-compose

Đặt vấn đề và Giải pháp Container hóa hệ thống Trong các hệ thống phân tán hiện đại, việc cấu hình và đồng bộ môi trường triển khai (Deployment Environment) luôn là một thách thức lớn. Hệ thống Secure Chat tích hợp đa dạng các thành phần công nghệ với các runtime và dependency khác nhau: Core Server chạy trên nền tảng Java Virtual Machine (JVM), cơ sở dữ liệu quan hệ MySQL, môi trường thực thi trí tuệ nhân tạo Ollama dựa trên các thư viện tính toán hiệu năng cao (C++/CUDA), và đặc biệt là giải pháp truyền thông đa phương tiện Jitsi Meet Stack vốn bao gồm nhiều thành phần vi dịch vụ phức tạp hoạt động dựa trên các giao thức mạng chuyên biệt (XMPP, WebRTC).

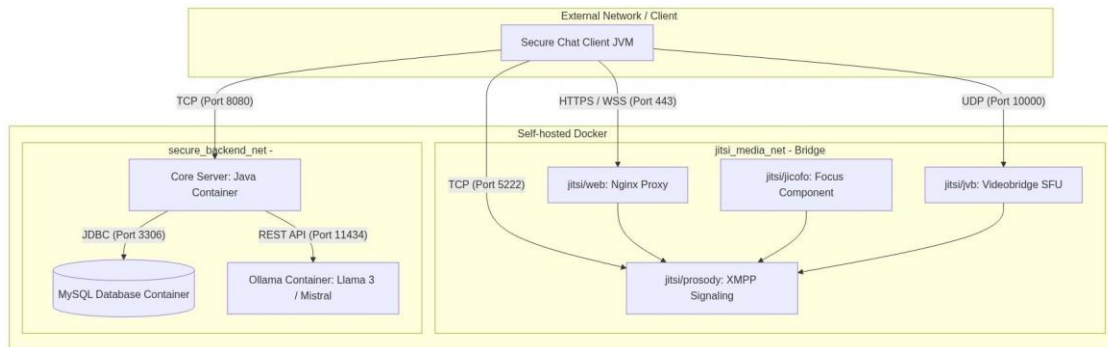
Việc triển khai các thành phần này theo phương pháp truyền thống (Bare-metal hoặc Virtual Machines) dẫn đến nhiều bất cập: xung đột thư viện hệ thống, quản lý tài nguyên kém linh hoạt, cấu hình mạng phức tạp và khó khăn trong việc mở rộng quy mô (Scaling). Nhằm tối ưu hóa quy trình triển khai Self-hosted độc lập, đảm bảo tính đóng gói (Encapsulation) và tính nhất quán trên mọi hạ tầng phần cứng, hệ thống áp dụng giải pháp công nghệ ảo hóa cấp độ hệ điều hành (OS-level virtualization) thông qua **Docker** và quản lý đa container bằng **Docker-compose**. Việc sử dụng Docker giúp cô lập hoàn toàn không gian runtime của từng dịch vụ, giảm thiểu overhead tài nguyên so với ảo hóa phần cứng (Hypervisor-based VM), đồng thời chuẩn hóa luồng giao tiếp mạng nội bộ giữa các dịch vụ cấu thành.

Thiết kế mô hình phân rã dịch vụ và Cấu trúc liên kết mạng Hệ thống được thiết kế theo mô hình kiến trúc hướng dịch vụ (Service-Oriented Architecture), phân rã toàn bộ hạ tầng thành các container độc lập và tự vận hành. Các container này được liên kết và quản lý tập trung thông qua một tệp cấu hình quy chuẩn docker-compose.yml. Để đảm bảo tính bảo mật và kiểm soát luồng dữ liệu nghiêm ngặt, hệ thống thiết lập hai mạng ảo nội bộ độc lập (Custom Docker Networks) sử dụng driver bridge:

`secure\\_backend\\_net`: Mạng nội bộ dành riêng cho tầng xử lý dữ liệu lõi, kết nối giữa Core Server, Cơ sở dữ liệu MySQL và công cụ xử lý Local AI (Ollama). Mạng này hoàn toàn cô lập, không public các công dịch vụ nhạy cảm như cơ sở dữ liệu hay API AI ra môi trường Internet bên ngoài, ngăn chặn tuyệt đối các nguy cơ tấn công dò quét cổng (Port Scanning).

`jitsi\\_media\\_net`: Mạng dành riêng cho Jitsi Meet Stack nhằm tối ưu hóa luồng truyền tải đa phương tiện thời gian thực. Các dịch vụ trong mạng này tương tác trực tiếp với nhau và mở các cổng giao tiếp UDP/TCP cụ thể để thiết lập kênh truyền thông WebRTC với ứng dụng Client.

Luồng liên kết kiến trúc và phân chia ranh giới mạng giữa các container cấu thành trong hệ thống được thể hiện chi tiết qua sơ đồ dưới đây:



Hình 8: Sơ đồ kiến trúc Containerization và phân vùng mạng nội bộ Docker

Phân tích chức năng và Cấu hình kỹ thuật của các thành phần Container Để hệ thống vận hành ổn định với hiệu năng cao, cấu hình của từng container trong tệp cấu hình docker-compose.yml được tinh chỉnh chuyên sâu, phân bổ tài nguyên và ràng buộc thứ tự khởi chạy thông qua cơ chế depends_on. Các đặc tính kỹ thuật cốt lõi của các container được chi tiết hóa trong bảng dưới đây:

Bảng 2.1: Đặc tả kỹ thuật và cấu hình các Container trong hệ thống

Tên Dịch vụ	Docker Image gốc	Cổng tiếp xúc	Cơ chế cấu hình \	Ràng buộc hạ tầng
db_server	mysql:8.0	Nội bộ: 3306	Áp dụng cơ chế volumes để lưu trữ dữ liệu bền vững (named volumes). Sử dụng biến môi trường để thiết lập mã hóa ký tự utf8mb4.	

Tên Dịch vụ	Docker Image gốc	Cổng tiếp xúc	Cơ chế cấu hình \	Ràng buộc hạ tầng
ollama_ai	ollama/ollama:latest	Nội bộ: 11434	Cấu hình cơ chế deploy.resources.reservations để truy cập trực tiếp vào phần cứng GPU (NVIDIA CUDA) của Host, tăng tốc độ suy luận mô hình.	
core_app	openjdk:17-slim	Công khai: 8080	Đóng gói mã nguồn ứng dụng lõi dưới dạng file JAR tối giản. Ràng buộc depends_on với điều kiện service_health từ container db_server.	
jitsi_web	jitsi/web:latest	Công khai: 443	Đóng vai trò là Reverse Proxy và Web Server chính. Phục vụ việc phối tập cấu hình và định tuyến các kết nối HTTPS/WSS ban đầu từ Client.	
jitsi_prosody	jitsi/prosody:latest	Công khai: 5222	Xử lý giao thức truyền tin XMPP, quản lý phiên và xác thực	

Tên Dịch vụ	Docker Image gốc	Cổng tiếp xúc	Cơ chế cấu hình \ quyền tham gia phòng họp của người dùng.	Ràng buộc hạ tầng
jitsi_jicofo	jitsi/jicofo:latest	Không (Nội bộ)	Đóng vai trò thực thi logic nghiệp vụ điều phối hội nghị, kết nối trực tiếp với Prosody để quản lý trạng thái phòng họp.	
jitsi_jvb	jitsi/jvb:latest	Công khai: 10000/UDP	Cấu hình biến môi trường DOCKER_HOST_ADDR để định danh chính xác IP công cộng của máy chủ, tối ưu luồng WebRTC thông qua định tuyến NAT trực tiếp.	

Cơ chế đồng bộ, Lưu trữ bền vững và Quản lý vòng đời Một trong những thách thức lớn khi triển khai hệ thống State-full (như cơ sở dữ liệu và mô hình AI nặng hàng Gigabyte) trên nền tảng Docker là quản lý trạng thái dữ liệu (State). Hệ thống giải quyết triệt để bài toán này thông qua hai giải pháp kỹ thuật cụ thể:

Quản lý vòng đời, cơ chế đồng bộ và lưu trữ bền vững: Việc vận hành các hệ thống có trạng thái (State-full) như cơ sở dữ liệu và mô hình AI dung lượng lớn trên Docker thường đối mặt với rủi ro mất mát dữ liệu. Để giải quyết triệt để vấn đề này, hệ thống Secure Chat áp dụng hai giải pháp kỹ thuật trọng tâm: **Vùng nhớ lưu trữ bền vững (Persistent Volumes):** Dữ liệu cốt lõi từ MySQL (`/var/lib/mysql`) và các trọng số mô hình LLM của Ollama (`/root/.ollama`) được Mapping trực tiếp với các *Named Volumes* trên máy Host. Cơ chế này đảm bảo toàn bộ cấu hình, dữ liệu AI và lịch sử hội thoại luôn được bảo toàn tuyệt đối khi container thực hiện cập nhật image hoặc khởi động lại.

Điều phối phụ thuộc và Kiểm tra sức khỏe (Healthchecks & Dependency Control): Nhằm đảm bảo Core Server chỉ hoạt động khi MySQL đã sẵn sàng cấu trúc bảng, hệ thống tích hợp lệnh `mysqladmin ping` trong block `healthcheck`. Thông qua thuộc tính `condition: service_healthy`, tiến trình khởi chạy được

cường bức theo tuần tự: Thiết lập hạ tầng AI và lưu trữ → Xác minh trạng thái → Kích hoạt dịch vụ Core → Khởi chạy Jitsi Meet Stack.

2.4. Nền tảng Jitsi Meet Stack (Prosody, Jicofo, JVB) cho Video Call

Giải pháp hội nghị truyền hình (Video/Audio Call) trong dự án Secure Chat được xây dựng trên nền tảng hạ tầng phân tán Jitsi Meet Stack độc lập (Self-hosted) thông qua các container Docker. Thay vì sử dụng kiến trúc kết nối ngang hàng thuần túy (Peer-to-Peer - P2P) vốn bị giới hạn băng thông và tài nguyên xử lý tại Client khi số lượng người tham gia tăng lên, hệ thống áp dụng kiến trúc SFU (Selective Forwarding Unit) đa luồng. Hạ tầng này được cấu thành từ ba thành phần máy chủ cốt lõi: Prosody, Jicofo, và Jitsi Videobridge (JVB).

1. Kiến trúc phân rã các thành phần trong Jitsi Meet Stack

Mỗi thành phần trong kiến trúc đóng một vai trò chuyên biệt và giao tiếp với nhau thông qua giao thức truyền tin báo hiệu hoặc định tuyến dòng dữ liệu thời gian thực:

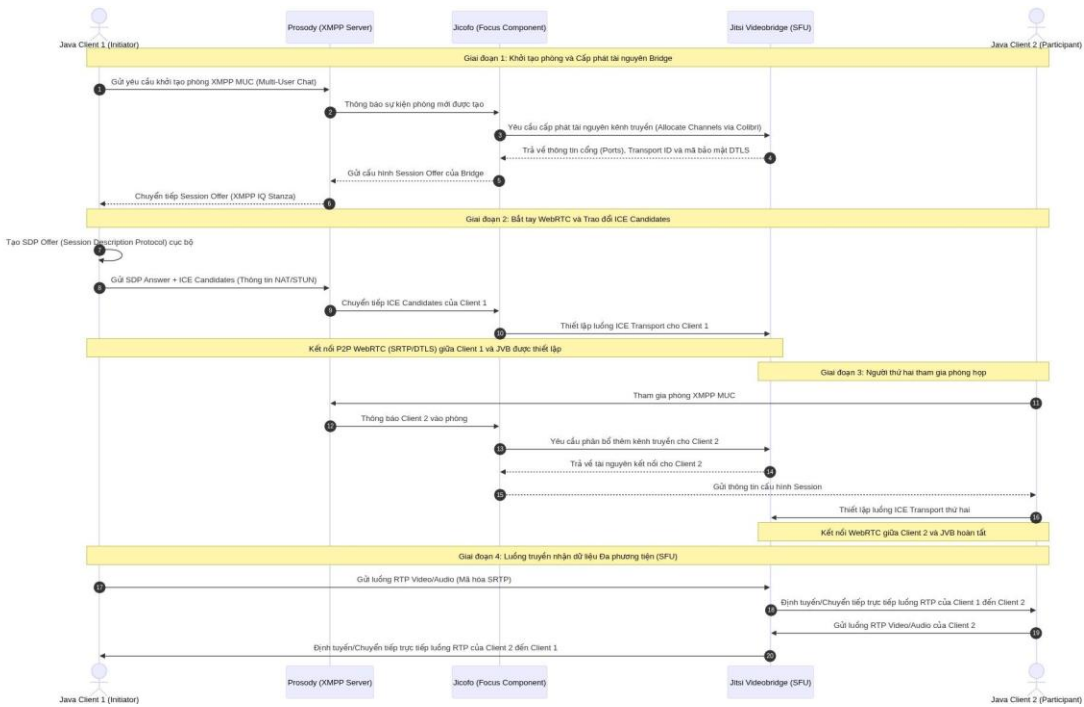
Prosody (XMPP Server): Đóng vai trò là nền tảng báo hiệu (Signaling Backbone) của toàn bộ hệ thống hội thoại. Prosody quản lý việc thiết lập kết nối, xác thực quyền truy cập của các Client, duy trì danh sách phòng họp trực tuyến và quản lý trạng thái hiện diện (Presence) thông qua giao thức mở rộng XMPP (Extensible Messaging and Presence Protocol). Tất cả thông tin điều phối phiên từ Client đều phải đi qua Prosody để phân phối đến các thành phần quản lý logic.

Jicofo (Jitsi Conference Focus): Là thành phần chịu trách nhiệm điều phối logic trung tâm (Conference Coordinator) của các phiên đa phương tiện. Jicofo không tham gia vào quá trình truyền nhận luồng media trực tiếp mà chỉ tập trung quản lý vòng đời của một phòng họp (Conference session lifecycle). Khi một Client yêu cầu khởi tạo hoặc tham gia cuộc gọi, Jicofo nhận tín hiệu từ Prosody, tính toán tài nguyên hiện tại của hạ tầng, sau đó thực hiện phân bổ và chỉ định một thực thể JVB phù hợp để phục vụ phòng họp đó thông qua giao thức nội bộ XEP-0343 (Signaling WebRTC Conferences in XMPP).

JVB (Jitsi Videobridge - SFU): Trái tim chịu tải đa phương tiện của hệ thống. Khác với kiến trúc MCU (Multipoint Control Unit) truyền thống vốn yêu cầu máy chủ phải giải mã, trộn tất cả các luồng video lại thành một luồng duy nhất rồi mã hóa lại để gửi về cho Client (gây tiêu tốn tài nguyên CPU máy chủ cực lớn và tăng độ trễ), JVB hoạt động theo cơ chế SFU. JVB nhận trực tiếp các luồng RTP/RTCP (Real-time Transport Protocol) chứa video/audio được mã hóa từ mỗi Client tham gia, sau đó chuyển tiếp (Forward) nguyên bản luồng dữ liệu đó đến tất cả các Client còn lại trong phòng mà không thực hiện bất kỳ thao tác giải mã hay xử lý đồ họa nào. Điều này tối ưu hóa tối đa hiệu suất CPU của Server, cho phép mở rộng băng thông xử lý đồng thời lên đến hàng trăm luồng stream.

2. Phân tích luồng thiết lập cuộc gọi và Định tuyến WebRTC

Quá trình khởi tạo và thiết lập một phiên Video Call giữa các Client Java Swing thông qua Jitsi Meet Stack tuân thủ nghiêm ngặt giao thức WebRTC (Web Real-Time Communication) phối hợp với hạ tầng XMPP. Luồng xử lý kỹ thuật chi tiết bao gồm các giai đoạn bắt tay và thiết lập kênh truyền được đặc tả qua sơ đồ tuần tự dưới đây:



Hình 9:: Luồng tuần tự thiết lập phiên kết nối WebRTC và định tuyến đa phương tiện qua hạ tầng Jitsi Meet Stack.

3. Cơ chế tối ưu hóa băng thông bằng giải thuật Simulcast và BWE

Để đảm bảo chất lượng cuộc gọi luôn ổn định trên ứng dụng Java Swing ngay cả trong điều kiện mạng của các Client bị biến động (Packet loss, nghẽn mạng), hệ thống Secure Chat tích hợp hai giải thuật tối ưu hóa băng thông lớp sâu của Jitsi:

Giải thuật Simulcast (Phát đa luồng độ phân giải): Khi Client gửi luồng video lên JVB, thay vì chỉ gửi một luồng duy nhất, Client sẽ đồng thời mã hóa và đẩy lên 3 luồng video có độ phân giải và bitrate khác nhau (ví dụ: High - 720p tại 2.5 Mbps, Medium - 360p tại 500 Kbps, và Low - 180p tại 150 Kbps). Dựa vào băng thông tải xuống hiện tại của từng Client nhận, JVB sẽ quyết định chuyển tiếp luồng có độ phân giải phù hợp nhất. Điều này ngăn chặn hiện tượng một Client có mạng yếu làm ảnh hưởng đến hiệu năng truyền tải của toàn bộ phòng họp.

Cơ chế ước lượng băng thông BWE (Bandwidth Estimation): JVB liên tục giám sát các gói tin RTCP (Real-time Control Protocol) phản hồi từ các Client để tính toán tỷ lệ mất gói (Packet Loss Rate) và độ trễ phản hồi (Round Trip Time - RTT). Hệ thống áp dụng giải thuật Google Congestion Control (GCC) hoạt động ở cả hai phía: dựa trên tổn thất (Loss-based) và dựa trên độ trễ (Delay-based) để dự đoán chính xác ngưỡng băng thông khả dụng tối đa, từ đó chủ động hạ hoặc nâng độ phân giải video theo thời gian thực mà không làm gián đoạn cuộc gọi.

4. Bảng thông số cấu hình hệ thống Jitsi Stack trong môi trường Docker

Các thông số hạ tầng triển khai trên Docker Self-hosted nhằm tối ưu hóa tải cho các thành phần dịch vụ được chi tiết hóa trong Bảng 1.

Thông số cấu hình kỹ thuật hạ tầng Jitsi Meet Stack

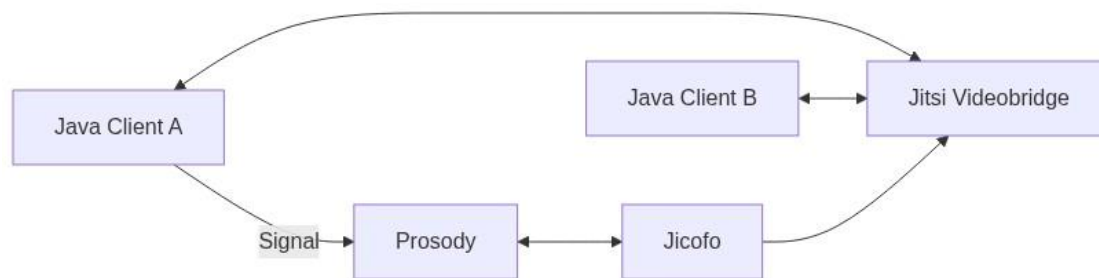
Thành phần máy chủ	Cấu hình / Tham số cốt lõi	Ý nghĩa kỹ thuật và Mục tiêu vận hành
Prosody	ENABLE_AUTH=true	
AUTH_TYPE=internal_hashed	Bắt buộc xác thực mọi kết nối Client thông qua tài khoản hệ thống Secure Chat trước khi cho phép tham gia máy chủ XMPP.	
Jicofo	JICOFO_MAX_MEMORY=1024m	
BRIDGE_SELECTION_STRATEGY=RegionBased	Giới hạn bộ nhớ phân bổ cho JVM của Jicofo; áp dụng chiến lược chọn JVB tối ưu theo khoảng cách địa lý và tải phần cứng.	
JVB (SFU Bridge)	VIDEOBRIDGE_MAX_MEMORY=2048m	
org.jitsi.videobridge.REST_SHUTDOWN_ENABLE=true	Cấp phát tài nguyên bộ nhớ đệm cho định tuyến gói tin RTP; bật API quản trị cho phép ngắt kết nối an toàn khi bảo trì.	
Mạng	Cổng kết nối	Cổng 10000/UDP (Luồng RTP/SRTP)
Cổng 443/TCP (Tín hiệu WebSockets/HTTPS)	Tách biệt luồng dữ liệu truyền thông đa phương tiện tốc độ cao (UDP) và luồng	

Thành phần máy chủ	Cấu hình / Tham số cốt lõi dữ liệu điều khiển báo hiệu an toàn (TCP).	Ý nghĩa kỹ thuật và Mục tiêu vận hành
WebRTC Security	ENABLE_DTLS _SRTP=true	
ENABLE_XMPP _WEBSOCKET=true	Kích hoạt mã hóa lớp truyền tải DTLS-SRTP, đảm bảo bảo mật tuyệt đối cho luồng video/audio từ Client tới JVB.	

2.3.1. Jitsi Meet Stack và Kiến trúc Containerized

Hệ thống tích hợp Jitsi Meet làm giải pháp hội nghị truyền hình cốt lõi. Trong môi trường Docker, Jitsi được phân rã thành nhiều microservices phối hợp nhịp nhàng.

Các thành phần chính trong Docker Compose: **Prosody (XMPP Server):** Đóng vai trò trung tâm điều phối tín hiệu và quản lý phiên. **Jicofo (Jitsi Conference Focus):** Quản lý việc thiết lập các phòng họp và phân bổ tài nguyên. **JVB (Jitsi Videobridge):** Thành phần quan trọng nhất (Selective Forwarding Unit - SFU), chịu trách nhiệm chuyển tiếp luồng video giữa các thành viên mà không cần giải mã, giúp giảm tải CPU cực lớn cho Server.



Hình 10: Kiến trúc luồng tín hiệu và luồng media trong Jitsi Meet.

Tổng quan về Local AI và Ollama Framework

Bản chất của Local AI và Bài toán Bảo mật thông tin Trong cấu trúc phát triển của các hệ thống phần mềm thông minh hiện nay, việc tích hợp các Mô hình Ngôn ngữ Lớn (Large Language Models - LLMs) đóng vai trò cốt lõi trong việc nâng cao trải nghiệm người dùng thông qua khả năng hiểu và xử lý ngôn ngữ tự nhiên. Tuy nhiên, kiến trúc tích hợp AI truyền thống dựa vào mô hình Điện toán đám mây (Cloud AI) thông qua các RESTful API thương mại (như OpenAI API, Anthropic Claude API) đang đối mặt với những thách thức nghiêm trọng về mặt an ninh thông tin. Khi ứng dụng Client hoặc Core Server gửi dữ liệu dạng tường minh (Plaintext) lên Cloud để xử lý suy luận (Inference), tổ chức hoàn toàn mất quyền kiểm soát đối với dữ liệu đó. Các rủi ro

bao gồm việc rò rỉ dữ liệu hội thoại nhạy cảm, vi phạm các tiêu chuẩn pháp lý (như GDPR, HIPAA) hoặc dữ liệu bị sử dụng trái phép để huấn luyện lại mô hình (Retraining).

Khái niệm **Local AI** (Trí tuệ nhân tạo cục bộ) ra đời như một giải pháp thay đổi kiến trúc (Architectural Shift), dịch chuyển toàn bộ tài nguyên tính toán và suy luận của mô hình LLM từ hạ tầng đám mây công cộng về trực tiếp hạ tầng máy chủ nội bộ (Self-hosted Server) của tổ chức. Bằng cách vận hành cục bộ, chu trình luân chuyển dữ liệu (Data Lifecycle) của một tác vụ AI được giới hạn hoàn toàn trong phân vùng mạng nội bộ. Điều này đảm bảo tính *Zero-Knowledge* đối với các thực thể bên ngoài, đồng thời loại bỏ sự phụ thuộc vào kết nối Internet và giảm thiểu rủi ro từ sự cố gián đoạn dịch vụ của bên thứ ba.

Công nghệ cốt lõi của Framework Ollama Để vận hành các mô hình LLM có tham số lớn (chẳng hạn như Llama 3 với 8 tỷ tham số) trên hạ tầng máy chủ thông thường mà vẫn đảm bảo hiệu năng cao, dự án tích hợp hệ thống **Ollama Framework**. Ollama là một công cụ mã nguồn mở được thiết kế để chuẩn hóa, đóng gói và tối ưu hóa quy trình chạy các mô hình mã nguồn mở trên các môi trường phần cứng khác nhau.

Sự tối ưu về mặt học thuật và kỹ thuật của Ollama dựa trên các nền tảng sau:

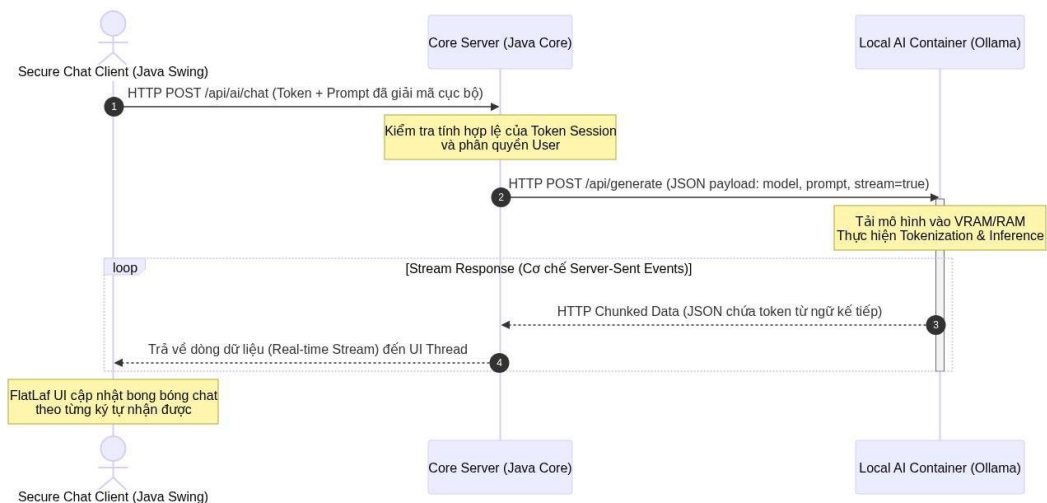
Phương thức Định lượng hóa (Quantization): Thông thường, các mô hình LLM được lưu trữ dưới định dạng dấu phẩy động 16-bit hoặc 32-bit (FP16/FP32), đòi hỏi tài nguyên VRAM/RAM rất lớn để tải toàn bộ trọng số. Để tối ưu hóa, Ollama áp dụng các thuật toán nén định lượng (điển hình là chuẩn $Q4_K_M$), chuyển đổi trọng số sang định dạng số nguyên 4-bit. Quy trình này giúp tinh gọn dung lượng bộ nhớ từ 16GB xuống dưới 5GB đối với dòng mô hình 8B, cho phép vận hành ổn định trên cấu hình phần cứng nội bộ mà vẫn duy trì chỉ số năng lực học thuật (Perplexity) ở mức tối ưu.

Nền tảng thực thi *llama.cpp*: Ollama vận hành dựa trên backend *llama.cpp* — một thư viện C/C++ tối giản giúp đẩy nhanh tốc độ tính toán ma trận trên CPU thông qua các tập lệnh vector hóa (AVX2, AVX-512) hoặc khai thác sức mạnh đồ họa GPU thông qua kiến trúc NVIDIA CUDA.

Quản trị mô hình qua Modelfile: Tương tự như cơ chế Dockerfile, Ollama cho phép định nghĩa cấu trúc mô hình và thiết lập các tham số vận hành như *temperature*, *top_k*, *top_p* cùng câu lệnh điều hướng hệ thống (System Prompt). Điều này giúp chuẩn hóa phong cách phản hồi của trợ lý ảo một cách minh bạch và chính xác.

Trong kiến trúc hệ thống Secure Chat, thành phần Ollama được triển khai biệt lập bên trong một Docker Container thuộc phân vùng mạng an toàn `secure\_backend\_net`. Quy trình xử lý suy luận Local AI đảm bảo tính bảo mật khi các yêu cầu từ Client (Java Swing) không kết nối trực tiếp với Ollama mà bắt buộc phải thông qua tầng trung gian Core Server để thực hiện xác thực và kiểm soát quyền truy cập.

Luồng xử lý kỹ thuật chi tiết của một tác vụ gọi trợ lý AI được mô tả tuần tự qua sơ đồ dưới đây:



Hình 11: Sơ đồ tuần tự (Sequence Diagram) luồng xử lý truy vấn Local AI qua Ollama

Để làm rõ cơ chế tương tác dữ liệu, bảng dưới đây mô tả chi tiết các tham số kỹ thuật đầu vào và đầu ra của cấu trúc API giữa Core Server và Ollama Framework:

Bảng 2.2: Đặc tả cấu trúc Payload API giao tiếp nội bộ với Ollama

Tham số	Kiểu dữ liệu	Vị trí / Loại	Giải thích ý nghĩa kỹ thuật
model	String	Request Body	Xác định định danh mô hình LLM cục bộ cần gọi (llama3:8b hoặc mistral:7b).
prompt	String	Request Body	Nội dung ngữ cảnh câu hỏi của người dùng sau khi đã được chuẩn hóa cấu trúc hội thoại.
options	JSON Object	Request Body	Chứa các siêu tham số điều khiển thuật toán sinh từ: temperature (độ sáng tạo), num_ctx (kích thước cửa sổ ngữ cảnh).
stream	Boolean	Request Body	Thiết lập bằng true để kích hoạt cơ chế truyền tải

Tham số	Kiểu dữ liệu	Vị trí / Loại	Giải thích ý nghĩa kỹ thuật dạng dòng (Streaming), tối ưu hóa thời gian phản hồi đầu tiên (Time to First Token - TTFT).
response	String	Response Chunk	Token từ ngữ đơn lẻ được sinh ra tiếp theo bởi mô hình trong quá trình suy luận ma trận xác suất.
done	Boolean	Response Chunk	Cờ đánh dấu trạng thái (Flag) thông báo mô hình đã hoàn tất chuỗi sinh từ (End of Sequence - EOS).

Qua cơ chế thiết kế tích hợp này, hệ thống Secure Chat vừa khai thác được sức mạnh học thuật của các mô hình ngôn ngữ lớn tiên tiến nhất thế giới, vừa duy trì được một kiến trúc bảo mật khép kín tuyệt đối, đáp ứng hoàn hảo tiêu chí cốt lõi của một hệ thống nhấn tin an toàn chuẩn doanh nghiệp.

2.5 Cơ chế tích hợp Ollama API vào hệ thống Chat (Local LLM prompt engineering)

Việc tích hợp trợ lý ảo AI chạy cục bộ (Local AI) vào hệ thống Secure Chat nhằm giải quyết triệt để bài toán bảo mật thông tin, ngăn chặn rò rỉ dữ liệu hội thoại nhạy cảm lên các dịch vụ đám mây bên thứ ba. Thành phần xử lý cốt lõi dựa trên framework Ollama chạy trong môi trường hạ tầng tự vận hành (Self-hosted Docker) của hệ thống, cung cấp các mô hình ngôn ngữ lớn (LLM) tối ưu như Llama 3 hoặc Mistral.

Dưới góc độ kiến trúc, Client Java Swing không tương tác trực tiếp với mô hình mà thông qua các luồng API không đồng bộ, kết hợp kỹ thuật kỹ nghệ định hướng (Prompt Engineering) chuyên sâu để chuẩn hóa dữ liệu đầu ra.

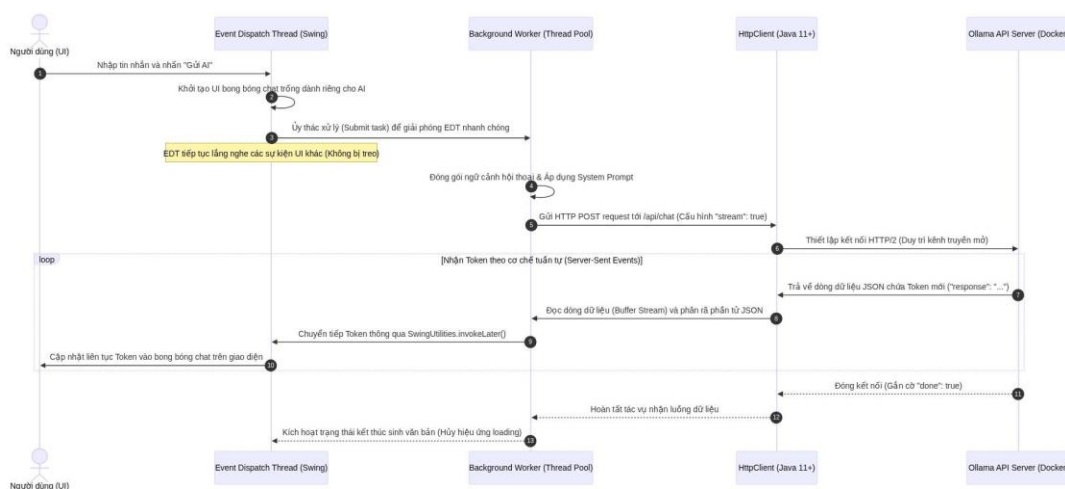
1. Kiến trúc kết nối không đồng bộ và Kỹ thuật truyền dữ liệu dạng dòng (Streaming API)

Để tích hợp Ollama API vào một ứng dụng giao diện trực quan như Java Swing, hệ thống bắt buộc phải giải quyết bài toán chặn luồng (Thread blocking). Do đặc thù thời gian suy luận (Inference time) của các dòng Local LLM phụ thuộc rất lớn vào năng lực tính toán của phần cứng (CPU/GPU) tại máy chủ nội bộ, thời gian tạo ra một câu trả lời hoàn chỉnh có thể kéo dài

từ vài giây đến hàng chục giây. Nếu Client thực hiện một cuộc gọi đồng bộ (Synchronous HTTP Request), luồng xử lý giao diện của Swing (Event Dispatch Thread - EDT) sẽ bị đóng băng, dẫn đến hiện tượng treo ứng dụng (ANR - Application Not Responding).

Do đó, hệ thống Secure Chat thiết lập cơ chế kết nối không đồng bộ sử dụng thư viện HttpClient của Java 11+ kết hợp với giao thức **Server-Sent Events (SSE)** để tiếp nhận dữ liệu trả về theo dạng dòng (*Streaming responses*) từ endpoint/api/generate hoặc/api/chat của Ollama. Kỹ thuật này giúp các token văn bản vừa được mô hình sinh ra sẽ lập tức truyền tải về Client và kết xuất (Render) lên bong bóng chat theo thời gian thực, tạo trải nghiệm mượt mà và giảm thiểu cảm giác chờ đợi của người dùng.

Luồng kiến trúc điều phối thông điệp và xử lý bất đồng bộ giữa các thành phần được mô tả chi tiết qua sơ đồ tuần tự dưới đây:



Hình 12: Sơ đồ tuần tự cơ chế truyền dữ liệu dạng dòng không đồng bộ từ Ollama API lên giao diện Java Swing.

2. Kỹ nghệ định hướng cấu trúc (Prompt Engineering) và Kiểm soát định dạng JSON

Đối với mô hình ngôn ngữ lớn cục bộ (Local LLM), thách thức lớn nhất là việc kiểm soát tính ổn định và định dạng của câu trả lời. Khác với các mô hình thương mại quy mô lớn (như GPT-4), các mô hình mã nguồn mở kích thước nhỏ (7B/8B tham số) dễ gặp hiện tượng “ảo tưởng” (Hallucination) hoặc trả về các chuỗi văn bản tự do, phá vỡ cấu trúc hiển thị của ứng dụng. Để giải quyết vấn đề này, hệ thống áp dụng kỹ thuật cấu trúc hóa Prompt (Prompt Structuring) kết hợp với các tham số cấu hình suy luận nghiêm ngặt tại máy chủ.

Mỗi yêu cầu gửi tới Ollama được đóng gói thành một cấu trúc phân cấp bao gồm ba thành phần Prompt chính:

System Prompt (Định hình vai trò): Đặt ở mức ưu tiên cao nhất nhằm quy định hành vi nền tảng của mô hình. Trong Secure Chat, System Prompt ép buộc mô hình hoạt động như một trợ lý bảo mật, định hình tông giọng chuyên nghiệp, ngắn gọn và bắt buộc phải hỗ trợ Markdown để FlatLaf render định dạng (bôi đậm, kẻ bảng, khối mã nguồn).

Few-Shot Prompting (Học qua ví dụ): Hệ thống tiêm (Inject) sẵn các cặp mẫu [Câu hỏi - Câu trả lời] chuẩn vào tiền ngữ cảnh nhằm hướng dẫn mô hình hiểu được cấu trúc câu trả lời mong muốn mà không cần huấn luyện lại (Fine-tuning).

Context Window Management (Quản lý cửa sổ ngữ cảnh): Do bộ nhớ RAM/VRAM của hệ thống Self-hosted có giới hạn, hệ thống triển khai giải thuật hàng đợi trượt (Sliding Window) để quản lý lịch sử trò chuyện. Chỉ có \$N\$ tin nhắn gần nhất (mặc định \$N=10\$) được giữ lại để làm ngữ cảnh đầu vào, ngăn chặn hiện tượng tràn cửa sổ ngữ cảnh (Context Overflow) và giảm thiểu tài nguyên tính toán (Compute resource) của Ollama.

Để đảm bảo tính nhất quán tuyệt đối khi hệ thống cần AI xử lý các tác vụ quản trị nội bộ (ví dụ: phân tích sắc thái tin nhắn, phân loại tệp tin độc hại), hệ thống kích hoạt chế độ format: "json" trong cấu hình API của Ollama, ép buộc mô hình sinh ra một chuỗi JSON hợp lệ theo Schema định sẵn để Client dễ dàng phân tách bằng thư viện Jackson.

3. Các tham số điều khiển siêu dữ liệu suy luận (Inference Hyperparameters)

Để cân bằng giữa tính sáng tạo và độ chính xác của trợ lý ảo AI, hệ thống triển khai bộ tham số điều khiển thuật toán lấy mẫu (Sampling algorithms) của LLM trong mỗi Request JSON gửi tới Ollama. Các tham số này được định cấu hình tối ưu chi tiết trong Bảng 1.

Cấu hình tham số siêu dữ liệu suy luận tối ưu cho Local LLM

Tham số cấu hình	Giá trị thiết lập	Giải trình bản chất thuật toán và Mục tiêu kỹ thuật
temperature	0.4	Kiểm soát mức độ ngẫu nhiên khi chọn Token. Giá trị thấp (0.4) giúp thu hẹp phân phối xác suất, ưu tiên các câu trả lời mang tính logic, chính xác và giảm thiểu hiện tượng nói sai sự thật.
top_k	40	Giới hạn tập hợp các từ tiếp theo có xác suất cao nhất xuống còn 40 từ. Loại bỏ hoàn toàn các token có xác suất cực thấp ở vùng đuôi (Long tail), giúp câu trả lời mạch lạc hơn.
top_p	0.9	Kỹ thuật lấy mẫu hạt nhân (Nucleus sampling). Chỉ chọn các token trong tập hợp có

Tham số cấu hình	Giá trị thiết lập	Giải trình bản chất thuật toán và Mục tiêu kỹ thuật
		tổng xác suất tích lũy đạt 90%. Tạo ra sự linh hoạt vừa đủ trong việc diễn đạt ngôn từ của AI mà không làm lệch hướng ngữ cảnh.
num_predict	512	Định nghĩa số lượng Token tối đa mô hình được phép sinh ra cho một câu trả lời (Max Tokens), giới hạn thời gian chiếm dụng tài nguyên GPU/CPU của một phiên Chat đơn lẻ.
repeat_penalty	1.2	Áp đặt hình phạt trọng số đối với các token đã xuất hiện trước đó, ngăn chặn triệt để lỗi vòng lặp vô hạn (Infinite loop text generation) thường gặp ở các mô hình cục bộ.

2.6 Local AI & LLM (Mô hình ngôn ngữ lớn chạy offline)

Xu hướng tích hợp AI vào ứng dụng nhắn tin đang bùng nổ, nhưng vấn đề quyền riêng tư luôn là rào cản. Secure Chat giải quyết vấn đề này bằng cách sử dụng **Local AI**.

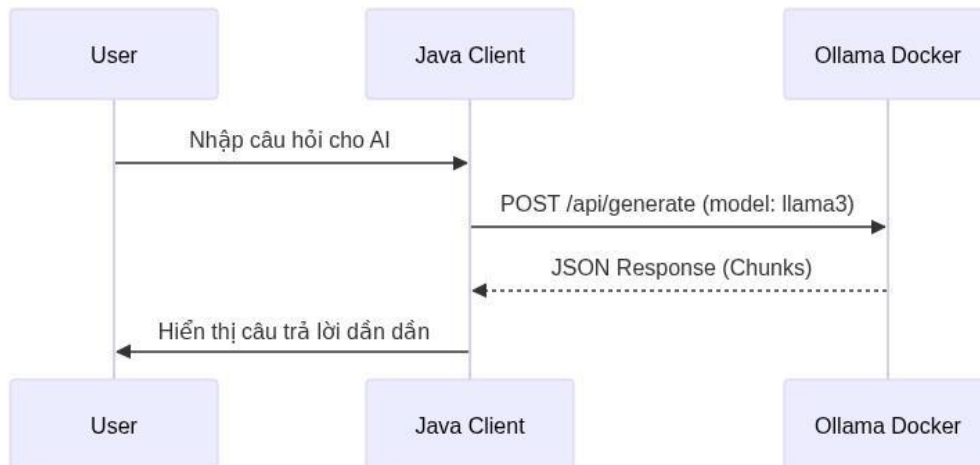
Lợi ích của việc chạy AI cục bộ: **Quyền riêng tư (Privacy):** Dữ liệu tin nhắn không bao giờ rời khỏi hạ tầng của người dùng. **Chi phí:** Không tốn phí API cho các bên thứ ba như OpenAI hay Google. **Khả năng hoạt động:** Có thể hoạt động trong mạng nội bộ (Intranet) không có Internet.

Hệ thống sử dụng kỹ thuật **Quantization (Lượng tử hóa)** để nén các mô hình khổng lồ xuống định dạng GGUF, cho phép chạy mượt mà trên các máy chủ có cấu hình vừa phải.

2.7 Framework Ollama (Cách tích hợp Llama 3 / Mistral vào Java)

Ollama được sử dụng làm cầu nối giữa các mô hình LLM mã nguồn mở và ứng dụng Java Swing.

Quy trình tích hợp: Hệ thống Java Client gửi yêu cầu HTTP POST đến Ollama Server (chạy trong Docker) qua cổng 11434. Dữ liệu được trao đổi dưới dạng JSON stream.



Hình 13: Luồng tương tác giữa Java Client và Trợ lý ảo AI.

2.8. Mã hóa đầu cuối (End-to-End Encryption - E2EE) hoạt động như thế nào?

Nguyên lý nền tảng của Cơ chế Mã hóa đầu cuối (E2EE) Mã hóa đầu cuối (End-to-End Encryption - E2EE) là một kiến trúc an ninh mạng mà trong đó chỉ có các thực thể tham gia giao tiếp trực tiếp (người gửi và người nhận) mới có khả năng đọc và giải mã thông điệp. Trong hệ thống Secure Chat, cơ chế E2EE triệt tiêu hoàn toàn vai trò rình mò hoặc thu thập dữ liệu bất hợp pháp từ các thực thể trung gian, bao gồm cả nhà cung cấp dịch vụ Internet (ISP) lẫn quản trị viên hệ thống lưu trữ tại máy chủ trung tâm (Server Platform).

Nguyên lý hoạt động cốt lõi tuân thủ chặt chẽ mô hình *Zero-Trust*: Server trung tâm đóng vai trò là một hub trung chuyển dữ liệu mù (Blind Router). Dữ liệu được mã hóa ngay tại bộ nhớ RAM của Client nguồn (Endpoint A) trước khi truyền qua mạng công cộng và chỉ được giải mã ngược lại tại bộ nhớ RAM của Client đích (Endpoint B). Toàn bộ chu trình luân chuyển dữ liệu trên đường truyền hay khi được lưu trữ trong hệ quản trị cơ sở dữ liệu MySQL đều tồn tại dưới dạng bản mã (Ciphertext) không thể đảo ngược nếu không có khóa bí mật tương ứng.

Mô hình Mã hóa lai (Hybrid Cryptosystem): Sự kết hợp giữa AES-256 và RSA-2048 Do đặc thù hiệu năng tính toán, các thuật toán mã hóa bất đối xứng (Asymmetric Encryption) như RSA có tốc độ xử lý rất chậm đối với các khối dữ liệu dung lượng lớn, nhưng lại có ưu thế tuyệt đối trong việc thiết lập kênh trao đổi khóa. Ngược lại, các thuật toán mã hóa đối xứng (Symmetric Encryption) như AES đạt hiệu năng xử lý cực cao, phù hợp cho việc truyền tin thời gian thực nhưng lại gặp bài toán học bùa về việc phân phối khóa an toàn. Hệ thống Secure Chat giải quyết bài toán tối ưu hóa hiệu năng bằng cách kết hợp cả hai giải pháp thành một mô hình Mã hóa lai (Hybrid Cryptosystem):

Dữ liệu khi đã chuyển thành bản mã (Ciphertext) sẽ trở nên bất khả xâm phạm và không thể khôi phục nếu thiếu khóa bí mật tương ứng.

Kiến trúc Mật mã lai (Hybrid Cryptosystem): Sự giao thoa giữa AES-256 và RSA-2048

Do những hạn chế về tốc độ xử lý khi đối mặt với khối lượng dữ liệu lớn của các thuật toán bất đối xứng (Asymmetric Encryption) như RSA, dù chúng sở hữu ưu thế vượt trội trong việc thiết lập kênh trao đổi khóa, hệ thống Secure Chat đã áp dụng mô hình mật mã lai để tối ưu hóa hiệu năng. Ngược lại với RSA, mã hóa đối xứng (Symmetric Encryption) như AES mang lại tốc độ xử lý tối ưu cho các tác vụ truyền tin thời gian thực nhưng lại gặp thách thức lớn trong việc phân phối khóa an toàn. Bằng cách kết hợp cả hai giải pháp, hệ thống đã hiện thực hóa mô hình Mã hóa lai với các tầng xử lý chuyên biệt:

Để khắc phục nhược điểm về tốc độ xử lý dữ liệu lớn của mã hóa bất đối xứng (RSA) và thách thức trong việc phân phối khóa của mã hóa đối xứng (AES), hệ thống Secure Chat đã triển khai mô hình Mật mã lai (Hybrid Cryptosystem). Giải pháp này tận dụng ưu thế của cả hai phương thức để tối ưu hóa hiệu năng truyền tin thời gian thực mà vẫn duy trì mức độ bảo mật cao nhất thông qua các giai đoạn chuyên biệt:

Nhằm khắc phục nhược điểm về tốc độ khi xử lý dữ liệu lớn của mã hóa bất đối xứng (RSA) và bài toán phân phối khóa phức tạp của mã hóa đối xứng (AES), hệ thống Secure Chat đã triển khai mô hình Mật mã lai (Hybrid Cryptosystem). Giải pháp này kết hợp ưu thế của cả hai phương thức để vừa tối ưu hóa hiệu năng truyền tải thời gian thực, vừa duy trì tính bảo mật tối đa qua các giai đoạn chuyên biệt:

- **Phân phối khóa an toàn (RSA-2048):** Sử dụng khóa công khai RSA 2048-bit của người nhận (KeyB), Client A thực hiện mã hóa khóa phiên session. Bản mã khóa key được tạo ra giúp quá trình chuyển giao tới Client B diễn ra an toàn, loại bỏ nguy cơ bị can thiệp bởi các thực thể trung gian.
- **Bảo mật và giải mã cục bộ (PBKDF2):** Tại máy trạm của Client B, khóa bí mật cá nhân (KeyB) được lưu trữ duy nhất trên thiết bị và bảo vệ bởi thuật toán băm PBKDF2. Khóa này chỉ được trích xuất khi cần giải mã bản mã khóa để khôi phục lại khóa phiên session nguyên bản.

Quy trình thực thi thông điệp mã hóa đầu cuối (E2EE)

Chu trình kỹ thuật từ mã hóa, truyền dẫn đến giải mã dữ liệu giữa hai đầu cuối được điều phối chặt chẽ thông qua hệ thống máy chủ trung tâm (Core Server). Sự tương tác đa tầng này cam kết tính bảo mật nội dung xuyên suốt từ người gửi tới người nhận. Dưới đây là mô tả chi tiết về luồng vận hành và các bước thực hiện tuần tự trong một phiên gửi thông điệp E2EE giữa Client A và Client B.

Bảo mật và giải mã tại máy trạm (PBKDF2): Khóa bí mật cá nhân (KeyB) được Client B lưu trữ duy nhất tại thiết bị và bảo vệ nghiêm ngặt bằng thuật toán băm PBKDF2. Khóa này chỉ được trích xuất để thực hiện tiến trình giải mã bản mã khóa, từ đó khôi phục lại khóa phiên session nguyên bản.

Tiến trình thực thi thông điệp mã hóa đầu cuối (E2EE)

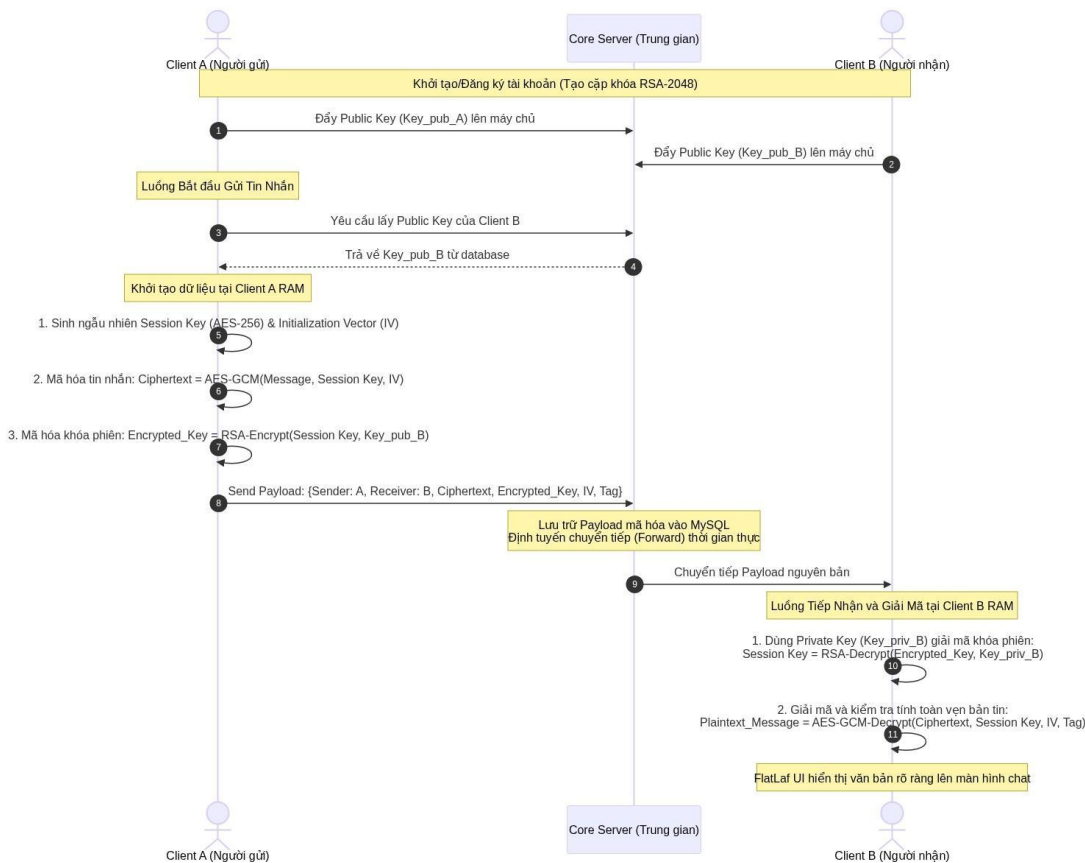
Quy trình kỹ thuật xử lý dữ liệu từ bước mã hóa, truyền dẫn đến giải mã giữa hai đầu cuối được điều phối chặt chẽ qua hệ thống máy chủ trung tâm (Core Server). Sự tương tác đa tầng này đảm bảo tính bảo mật nội dung xuyên suốt từ người gửi đến người nhận. Chi tiết về luồng vận hành

kỹ thuật và các bước thực thi tuần tự trong một phiên gửi tin nhắn E2EE giữa Client A và Client B được mô tả cụ thể qua sơ đồ trình tự (Sequence Diagram) dưới đây:

Quy trình kỹ thuật xử lý thông điệp E2EE

Luồng thực thi từ giai đoạn mã hóa, truyền dẫn đến giải mã giữa hai đầu cuối được thiết lập theo một quy trình kỹ thuật chặt chẽ thông qua hệ thống máy chủ trung tâm (Core Server). Sự phối hợp tương tác đa tầng trong chu trình gửi và nhận thông điệp E2EE, đảm bảo tính bảo mật xuyên suốt từ Client A đến Client B qua trung gian Server, được đặc tả và minh họa chi tiết thông qua sơ đồ trình tự (Sequence Diagram) sau đây.

Luồng vận hành kỹ thuật chi tiết của một phiên gửi tin nhắn E2EE Chu trình mã hóa, truyền tải và giải mã một thông điệp từ Client A đến Client B thông qua trung gian Core Server tuân thủ nghiêm ngặt theo các bước kỹ thuật tuần tự. Luồng tương tác phức tạp này được đặc tả chi tiết qua sơ đồ Sequence Diagram dưới đây:



Hình 14: Sơ đồ luồng xử lý và phân phối khóa kỹ thuật của giao thức mã hóa E2EE

Để làm rõ cấu trúc cấu thành của gói tin truyền tải trên đường truyền mạng giữa Client và Server, định dạng dữ liệu (Data Schema) được chuẩn hóa dưới dạng JSON Payload như mô tả chi tiết trong bảng dưới đây:

Bảng 2.3: Đặc tả cấu trúc gói tin mã hóa đầu cuối (E2EE Packet JSON)

Trường dữ liệu sender_id	Kiểu dữ liệu String	Vai trò và Ý nghĩa kỹ thuật trong mật mã học Định danh duy nhất (UUID) của người gửi phục vụ bài toán định tuyến.
recipient_id	String	Định danh duy nhất của người nhận để máy chủ xác định đích phát gói tin.
encrypted_msg	String	Bản mã của tin nhắn (Ciphertext) sau khi qua bộ lọc mã hóa AES-256-GCM, được mã hóa dưới dạng chuỗi Base64 để tránh lỗi ký tự hệ thống.
encrypted_key	String	Khóa phiên (Session Key) đã được mã hóa bằng khóa RSA-2048 Public Key của người nhận, biểu diễn dưới dạng Base64.
crypto_iv	String	Vector khởi tạo (Initialization Vector - IV) có độ dài 96 bits (12 bytes) ngẫu nhiên, đảm bảo không có hai bản mã nào trùng nhau kể cả khi nội dung tin nhắn gốc giống nhau.
auth_tag	String	Thẻ xác thực dữ liệu (Authentication Tag - MAC) độ dài 128 bits sinh ra bởi chế độ GCM, dùng để đầu nhận đối chiếu tính toàn vẹn, chống tấn công sửa đổi bit.

Cơ chế Toán học và Thuật toán thực thi Về mặt toán học, luồng xử lý mã hóa lai được thiết lập dựa trên các phương trình biến đổi đại số logic phức tạp. Giả sử thông điệp rõ là chuỗi bit M , các bước biến đổi toán học tại Client người gửi tuân thủ hai công thức độc lập:

$$C_{msg}, T = \text{extAES-GCM}(K_{session}, IV, M)$$

$$C_{key} = K_{session}^e \pmod n$$

Trong đó, cặp giá trị (e, n) là các hằng số toán học cấu thành nên khóa công khai RSA của người nhận (Key_{pub}).

Tại Client người nhận, quá trình giải mã khôi phục khóa phiên đối xứng được thực thi bằng cách tính toán hàm lũy thừa modulo với số mũ bí mật d (Key_{priv}):

$$K_{session} = C_{key}^d \pmod n$$

Sau khi khôi phục thành công giá trị của $K_{session}$, bản rõ ban đầu được giải mã hoàn chỉnh và xác thực thông qua hàm ngược của hệ thống mật mã đối xứng:

Nếu bất kỳ một bit nào của C_{msg} hoặc T bị thay đổi trên đường truyền mạng do nhiễu hoặc do sự can thiệp có chủ đích của tin tặc, thuật toán giải mã tại hàm (2.4) sẽ ngay lập tức ném ra ngoại lệ ứng dụng (AEADBadTagException). Client nhận sẽ hủy bỏ gói tin và phát cảnh báo an ninh lên giao diện người dùng, đảm bảo tính an toàn tuyệt đối cho toàn bộ hệ thống liên lạc.

2.9. Quản lý Session và Thuật toán mã hóa khóa (Sử dụng AES-256 và RSA-2048 trong Java)

Cơ chế quản lý phiên (Session Management) và bảo mật dữ liệu của hệ thống Secure Chat dựa trên mô hình mã hóa lai (Hybrid Cryptosystem), kết hợp giữa tính an toàn cao của mã hóa bất đối xứng (Asymmetric Cryptography) và hiệu năng xử lý tốc độ cao của mã hóa đối xứng (Symmetric Cryptography). Sự kết hợp này giải quyết triệt để bài toán phân phối khóa an toàn qua môi trường mạng không tin cậy mà vẫn đảm bảo tốc độ truyền tải tin nhắn thời gian thực trên ứng dụng Client Java Swing.

1. Thiết lập cặp khóa RSA-2048 và Cơ chế bảo vệ Private Key nội bộ

Mỗi tài khoản người dùng khi khởi tạo trên hệ thống sẽ được Client tự động sinh một cặp khóa RSA với độ dài kích thước *2048extbits* thông qua lớp `KeyPairGenerator` của thư viện mã hóa Java Cryptography Extension (JCE).

Public Key (Khóa công khai): Sau khi khởi tạo, Khóa công khai sẽ được chuyển đổi sang định dạng mã hóa X.509 (mã hóa Base64) và đẩy lên máy chủ trung tâm MySQL. Bất kỳ người dùng nào trong hệ thống cũng có thể truy vấn Public Key này để thiết lập kênh giao tiếp an toàn với chủ sở hữu.

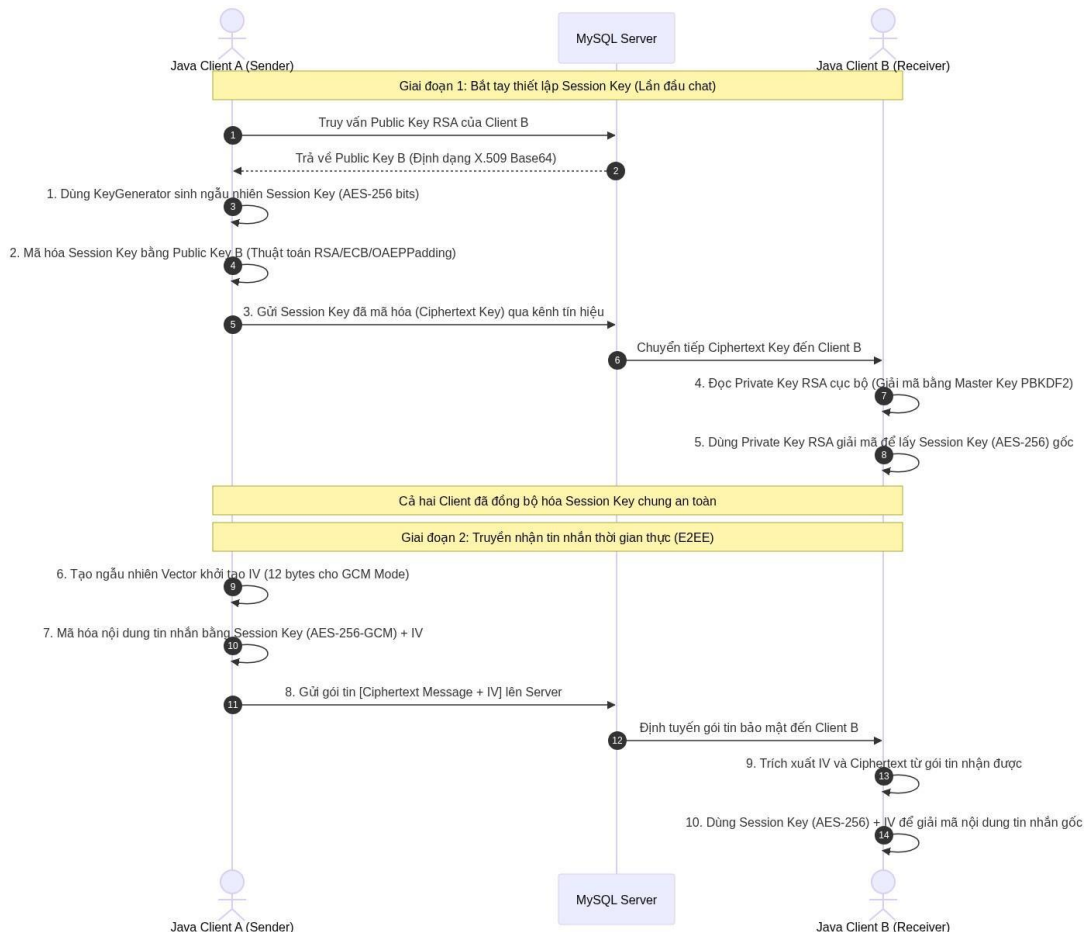
Private Key (Khóa bí mật) đóng vai trò then chốt đối với tính bảo mật của toàn bộ hạ tầng mã hóa đầu cuối (E2EE). Để đảm bảo an toàn tuyệt đối, hệ thống quy định nghiêm ngặt rằng khóa này không bao giờ được phép truyền tải qua mạng hoặc lưu trữ dưới dạng văn bản rõ (Plaintext) trên thiết bị. Nhằm bảo vệ Private Key tại máy trạm, giải pháp áp dụng thuật toán hàm dẫn xuất khóa PBKDF2 (kết hợp HmacSHA256) với cấu hình 10.000 vòng lặp và một chuỗi muối (Salt) ngẫu nhiên có kích thước 16 bytes.

Mật khẩu đăng nhập của người dùng được đưa vào hàm băm này để tạo ra một khóa mã hóa nền tảng (Master Key). Khóa nền tảng này sau đó được sử dụng làm khóa đối xứng cho thuật toán AES/CBC/PKCS5Padding nhằm mã hóa chính Khóa bí mật RSA trước khi lưu xuống bộ nhớ thiết bị dưới dạng tệp tin cấu hình. Quy trình này đảm bảo rằng ngay cả khi kẻ tấn công chiếm được quyền kiểm soát vật lý thiết bị Client, chúng vẫn không thể đọc được Private Key nếu không có mật khẩu gốc của người dùng.

2. Cơ chế quản lý Phiên (Session Key) và Luồng giao tiếp lai (Hybrid Encryption)

Hệ thống Secure Chat không sử dụng trực tiếp RSA để mã hóa từng tin nhắn vì chi phí tính toán số học lớn trên các số nguyên tố khổng lồ của RSA sẽ gây nghẽn tài nguyên CPU khi chat liên tục. Thay vào đó, RSA chỉ đóng vai trò bảo mật cho pha bắt tay trao đổi một Khóa phiên đối xứng ngẫu nhiên (Session Key) dạng AES-256 cho từng phiên trò chuyện (Chat Session).

Luồng kỹ thuật mã hóa và giải mã tin nhắn E2EE giữa Client A (Người gửi) và Client B (Người nhận) được thực thi tuần tự theo các bước khoa học dưới đây:



Hình 15: Sơ đồ tuần tự luồng bắt tay lai (Hybrid Handshake) và truyền nhận tin nhắn mã hóa E2EE qua JCE.

3. Hiện thực thuật toán mã hóa AES-256 ở chế độ GCM trong Java

Hệ thống Secure Chat bắt buộc sử dụng chế độ mã hóa **GCM (Galois/Counter Mode)** cho thuật toán AES-256 thay vì chế độ CBC truyền thống. Điểm ưu việt mang tính học thuật của AES-GCM là nó thuộc nhóm giải thuật mã hóa xác thực mã dòng (Authenticated Encryption with Associated Data - AEAD).

Trong quá trình mã hóa, AES-GCM tự động sinh ra một thẻ xác thực kèm theo (Authentication Tag - độ dài mặc định *128extbits*). Khi Client nhận thực hiện giải mã, thư viện JCE sẽ tính toán lại thẻ này; nếu dữ liệu tin nhắn bị kẻ tấn công can thiệp hoặc sửa đổi dù chỉ *1extbit* trên đường truyền, quá trình giải mã sẽ lập tức thất bại và ném ra ngoại lệ `AEADBadTagException`, ngăn chặn hoàn toàn các cuộc tấn công thay đổi nội dung (Bit-flipping attacks).

Các tham số lập trình đặc tả thuật toán trong thư viện JCE Java được quy định cụ thể tại Bảng 1.

Đoạn mã giả lập dưới đây minh họa logic toán học lỗi được lập trình ở tầng nghiệp vụ (Service layer) của Client Java Swing khi thực hiện mã hóa tin nhắn bằng AES-256-GCM:

```
// Khởi tạo Vector khởi tạo ngẫu nhiên (Secure IV) byte[] iv = new byte[12]; SecureRandom
random = new SecureRandom(); random.nextBytes(iv);

// Thiết lập cấu hình tham số mã hóa cho chế độ GCM Cipher cipher =
Cipher.getInstance("AES/GCM/NoPadding"); GCMParameterSpec spec = new
GCMParameterSpec(128, iv); cipher.init(Cipher.ENCRYPT_MODE, sessionKey, spec);

// Thực thi mã hóa dữ liệu văn bản rõ (Plaintext) byte[] cipherText =
cipher.doFinal(plainText.getBytes(StandardCharsets.UTF_8));

// Đóng gói mảng byte [IV (12B) + Ciphertext (NB)] trước khi chuyển đổi sang Base64 byte[]
encryptedPayload = ByteBuffer.allocate(iv.length + cipherText.length) .put(iv) .put(cipherText)
.array(); String finalPayloadBase64 = Base64.getEncoder().encodeToString(encryptedPayload);
```

Cơ chế đóng gói gộp chung Vector khởi tạo (IV) vào phần đầu của chuỗi byte dữ liệu gửi đi giúp Client nhận dễ dàng tách lọc (Parse) cấu trúc thông tin để đưa vào hàm khởi tạo giải mã, tối giản hóa việc lưu trữ trạng thái phiên trên Database của máy chủ trung gian.

Chương 3: ĐẶC TẢ YÊU CẦU PHẦN MỀM (SRS) và THIẾT KẾ HỆ THỐNG (SDD & OOD)

3.1. Danh sách tác nhân (Actors trong hệ thống)

Phân tích đặc điểm tác nhân và Ranh giới hệ thống

Trong quá trình phân tích và thiết kế hệ thống theo phương pháp hướng đối tượng (Object-Oriented Analysis and Design - OOAD), việc định nghĩa chính xác danh sách tác nhân (Actors) là bước cơ bản giúp xác định ranh giới hệ thống (System Boundary) và các tương tác hành vi. Một tác nhân đại diện cho một vai trò (Role) mà người dùng hoặc một hệ thống ngoại vi đảm nhận khi tương tác trực tiếp với hệ thống Secure Chat.

Do đặc thù hệ thống sử dụng kiến trúc phân tán Self-hosted, tích hợp mô hình tính toán cục bộ (Local AI) và hạ tầng truyền thông WebRTC đa thành phần, các tác nhân không chỉ giới hạn ở người dùng là con người (Human Actors) mà còn bao gồm các hệ thống phần mềm tự động (System/Non-human Actors). Các tác nhân này vận hành trong các phân vùng đặc quyền khác nhau, được phân tách rõ ràng để đảm bảo tính an ninh và nguyên tắc đặc quyền tối thiểu (Principle of Least Privilege).

Đặc tả chi tiết các Tác nhân trong hệ thống

Hệ thống Secure Chat xác định 04 tác nhân cốt lõi tham gia vào các chu trình vận hành nghiệp vụ. Chi tiết về bản chất kỹ thuật, mục tiêu tương tác và ranh giới bảo mật của từng tác nhân được đặc tả như sau:

Người dùng thông thường (Chat User): là tác nhân con người tương tác với hệ thống thông qua ứng dụng Client viết bằng Java Swing (được tối ưu hóa giao diện bằng thư viện FlatLaf). Tác nhân này nắm giữ cặp khóa mã hóa RSA-2048 cá nhân được lưu trữ dưới dạng mã hóa PBKDF2 tại bộ nhớ cục bộ trạm làm việc.

Phạm vi tương tác: Thực hiện các ca sử dụng (Use Cases) liên quan đến thiết lập hội thoại E2EE, gửi nhận gói tin mã hóa đối xứng, truyền tệp an toàn thông qua cơ chế phân phối FileTransferKey, khởi tạo hoặc tham gia các phiên gọi video thời gian thực dựa trên hạ tầng Jitsi Meet.

Ranh giới an ninh: Không có quyền can thiệp vào cấu hình hệ thống, cơ sở dữ liệu trung tâm hoặc tài nguyên phần cứng của máy chủ, giao tiếp với Core Server hoàn toàn qua cổng API được chứng thực bằng JSON Web Token (JWT).

Quản trị viên hệ thống (System Administrator): là tác nhân con người chịu trách nhiệm quản lý, giám sát và vận hành toàn bộ hạ tầng kỹ thuật của dự án.

Phạm vi tương tác: Tác nhân này tương tác chủ yếu ở tầng Server, thực thi các tác vụ quản lý vòng đời Container thông qua Docker-compose, cấu hình các tham số mạng ảo (secure_backend_net, jitsi_media_net), giám sát trạng thái lưu trữ của MySQL, phân quyền tài khoản và điều phối phân bổ tài nguyên phần cứng cho dịch vụ Local AI. Ranh giới an ninh: Nắm giữ đặc quyền tối cao đối với hạ tầng (Infrastructure Level) nhưng tuân thủ nghiêm ngặt mô hình Zero-Knowledge. Do cơ chế mã hóa đầu cuối (E2EE), quản trị viên hoàn toàn không thể tiếp cận khóa Private Key của người dùng, đồng nghĩa với việc không có khả năng đọc rõ nội dung tin nhắn hay tệp tin lưu trữ trong MySQL Database.

Hệ thống Trợ lý ảo cục bộ (Local AI Engine - Ollama): là một tác nhân hệ thống (System Actor)

đóng vai trò một thực thể tự động cung cấp dịch vụ suy luận ngôn ngữ tự nhiên.

Phạm vi tương tác: Tiếp nhận các Prompt yêu cầu từ Core Server qua giao thức HTTP REST API tại cổng nội bộ `11434`, thực thi tính toán ma trận xác suất dựa trên trọng số mô hình đã được định lượng hóa (Quantized Llama 3 / Mistral) và trả dữ liệu về dạng dòng (Stream) thông qua cơ chế Server-Sent Events (SSE).

Ranh giới an ninh: Nằm hoàn toàn trong vùng mạng cô lập `secure\\_backend\\_net`, không có quyền truy cập Internet ra ngoài và không tương tác trực tiếp với Client, mọi dữ liệu đầu vào đều phải được lọc và định tuyến qua Core Server.

Hệ thống truyền thông đa phương tiện (Jitsi Media Stack): là tác nhân hệ thống bao gồm tổ hợp các dịch vụ vi mô viễn thông tự động (Prosody, Jicofo, JVB).

Phạm vi tương tác: Thực thi tự động các giao thức báo hiệu (Signaling), quản lý trạng thái phòng họp ảo và định tuyến luồng dữ liệu thô (Audio/Video Streams) giữa các Client theo kiến trúc SFU WebRTC.

Ranh giới an ninh: Hoạt động độc lập trong phân vùng `jitsi\\_media\\_net`, mở tiếp xúc các cổng mạng UDP/TCP ra Internet để thiết lập kênh truyền thông trực tiếp với Client sau khi được Core Server xác thực quyền truy cập phiên họp.

Để tổng hợp và so sánh ranh giới quyền hạn cũng như cách thức tương tác kỹ thuật của các tác nhân, cấu trúc phân cấp được chi tiết hóa trong bảng dưới đây:

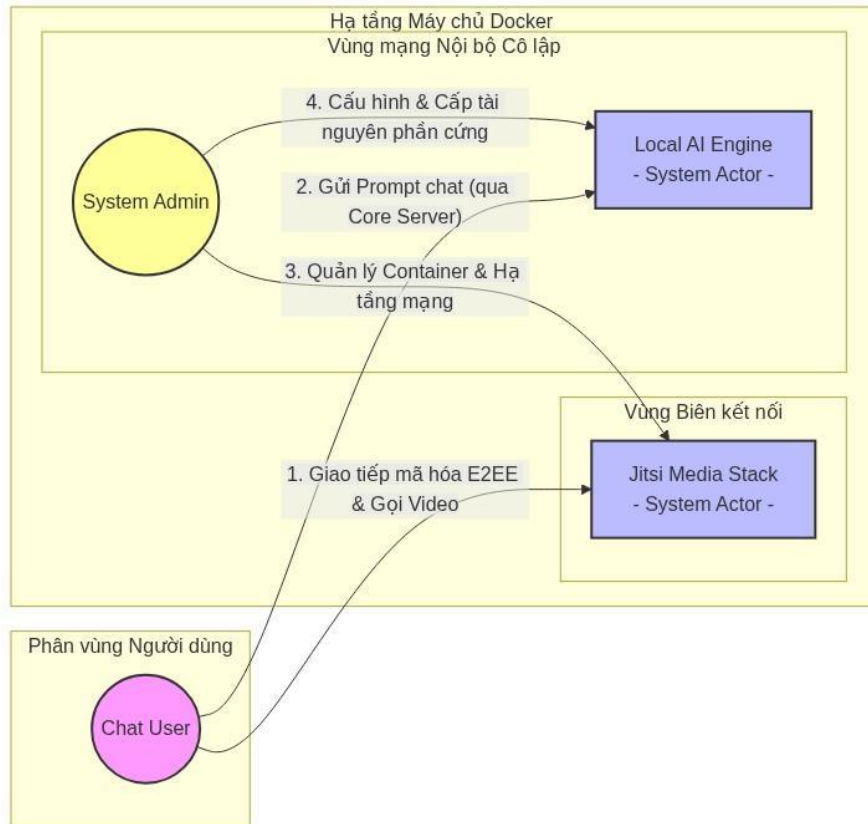
Bảng 3.1: Ma trận đặc tả và phân cấp đặc quyền của các Tác nhân

Tên Tác nhân	Loại Tác nhân	Giao thức tương tác chính	Giới hạn đặc quyền dữ liệu (Data Privacy Boundary)
Chat User	Con người (Human)	GUI Java Swing (FlatLaf) / HTTPS, WSS, WebRTC	Nắm giữ toàn quyền đối với khóa Private Key cá nhân và dữ liệu bản rõ hội thoại của chính mình tại vùng nhớ RAM Client. Không có quyền đọc dữ liệu của người dùng khác trên Server.

Tên Tác nhân	Loại Tác nhân	Giao thức tương tác chính	Giới hạn đặc quyền dữ liệu (Data Privacy Boundary)
System Admin	Con người (Human)	CLI (SSH, Docker-compose), SQL Scripts	Có quyền đọc/ghi toàn bộ hệ thống file cấu hình, bảng cơ sở dữ liệu thô, nhưng chỉ tiếp cận được dữ liệu hội thoại đã mã hóa dưới dạng Ciphertext Base64.
Local AI Engine	Hệ thống (System)	REST API (JSON payload) qua cổng nội bộ 11434	Chỉ xử lý dữ liệu ngữ cảnh ngắn hạn do Core Server gửi sang trong phiên suy luận đơn lẻ, không lưu giữ lịch sử hội thoại lâu dài vào đĩa cứng.
Jitsi Media Stack	Hệ thống (System)	XMPP (Port 5222), RTP/RTCP qua UDP (Port 10000)	Chỉ điều phối và định tuyến các luồng bitstream đa phương tiện thời gian thực, dữ liệu truyền thông không bị ghi lại trên ổ đĩa máy chủ (Non-persistent Media).

Sơ đồ mối quan hệ và Ranh giới tương tác của Tác nhân

Mối tương tác đa chiều giữa các tác nhân con người và tác nhân hệ thống vượt qua ranh giới bảo mật (Security Boundaries) để thực thi các chức năng cốt lõi của Secure Chat được mô tả tổng quát thông qua sơ đồ kiến trúc tương tác dưới đây:



Hình 16: Sơ đồ ranh giới và luồng tương tác giữa các Tác nhân trong kiến trúc hệ thống

3.2. Biểu đồ Use Case tổng quát

Biểu đồ Use Case tổng quát đóng vai trò đặc tả ranh giới hệ thống (System Boundary) và định hình toàn bộ các mối quan hệ tương tác giữa các tác nhân bên ngoài (Actors) với các chức năng nội tại của hệ thống Secure Chat. Dưới góc độ kiến trúc phần mềm, hệ thống được thiết kế theo mô hình phân rã chức năng cốt lõi nhằm đảm bảo tính cô lập và phân quyền nghiêm ngặt giữa bốn phân hệ chính: Xác thực (Auth), Nhắn tin bảo mật (Chat E2EE), Truyền thông đa phương tiện (Media Video Call) và Trí tuệ nhân tạo cục bộ (Local AI).

1. Xác định các Tác nhân (Actors) trong hệ thống

Kiến trúc phân quyền của Secure Chat định nghĩa hai nhóm tác nhân chính tham gia vận hành và sử dụng hệ thống:

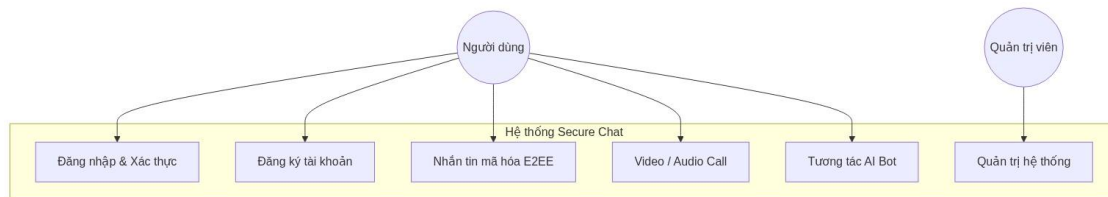
User (Người dùng ứng dụng Client): Tác nhân chủ thể chính tương tác với giao diện Java Swing. User sau khi vượt qua bộ lọc xác thực sẽ được cấp quyền truy cập toàn bộ các dịch vụ mã hóa đầu cuối bao gồm gửi tin nhắn, truyền tệp tin mã hóa, khởi tạo cuộc gọi WebRTC và truy vấn trợ lý ảo AI.

System Administrator (Quản trị viên hạ tầng): Tác nhân kỹ thuật tương tác thông qua CLI hoặc các bảng điều khiển quản trị của hạ tầng Docker Self-hosted. Người quản trị chịu trách nhiệm cấu hình các thông số hệ thống, giám sát trạng thái container lưu trữ dữ liệu MySQL, bảo trì hạ

tầng SFU Jitsi Videobridge, và kiểm soát chu kỳ hoạt động của các tiến trình cục bộ Ollama LLM.

2. Biểu đồ Use Case tổng quát hệ thống

Để mô tả mối quan hệ phụ thuộc giữa các Use Case thông qua các quan hệ kế thừa (*Generalization*), mở rộng (*Extend*) và bao hàm (*Include*), biểu đồ Use Case tổng quát được thiết kế cấu trúc lớp chức năng chi tiết qua sơ đồ dưới đây:



Hình 17: Biểu đồ Use Case tổng quát tích hợp các phân hệ nghiệp vụ hệ thống Secure Chat.

3. Phân tích phân rã kiến trúc kỹ thuật của các Use Case chính

Mối quan hệ logic trong kiến trúc Use Case của hệ thống không dừng lại ở mức biểu diễn hành vi mà liên kết chặt chẽ với các giải thuật toán học và luồng xử lý hạ tầng ngầm:

Mối quan hệ mật thiết giữa Xác thực và Quản lý Khóa: Khi người dùng thực hiện Đăng nhập & Xác thực hệ thống (Auth), hệ thống bắt buộc phải thực thi Use Case Khởi tạo cặp khóa RSA & KDF Private Key (KeyGen) theo quan hệ `<<include>>`. Ở bước này, Client Java Swing lấy mật khẩu của User chạy qua hàm băm `'PBKDF2WithHmacSHA256'` để làm khóa giải mã Private Key RSA đang lưu trữ cục bộ dưới dạng Ciphertext. Nếu giải mã thất bại, phiên làm việc sẽ bị hủy bỏ ngay lập tức, đảm bảo tính toàn vẹn của cơ chế E2EE.

Cơ chế bao hàm của Chuyển File an toàn: Use Case Chuyển tệp tin an toàn (SendFile) kế thừa cấu trúc truyền tin bảo mật của Chat. Tuy nhiên, để đảm bảo mỗi tệp tin đẩy lên Server Docker không thể bị đọc trộm bởi Admin máy chủ, SendFile bắt buộc phải gom cụm GenFileKey để sinh một khóa AES-256 đối xứng độc lập (FileTransferKey), thực hiện mã hóa tệp tin tại bộ nhớ tạm của Client trước khi đẩy luồng byte lên mạng.

Sự phụ thuộc báo hiệu WebRTC của phân hệ Media: Để thực thi Use Case Tham gia phòng WebRTC (JoinRoom), Client phải gửi token xác thực được cấp từ Auth sang cho máy chủ XMPP Prosody. Sau khi Prosody xác thực thành công, Jicofo mới tiến hành cấp phát tài nguyên luồng trên Jitsi Videobridge (JVB) nhằm kích hoạt tính năng Video Call đa luồng cho Client.

4. Bảng đặc tả chi tiết dòng sự kiện của các Use Case cốt lõi

Nhằm cụ thể hóa các điều kiện biên và kịch bản thực tế khi thực thi, Bảng 1 cung cấp tài liệu đặc tả cấu trúc của các Use Case đại diện cho các phân hệ chức năng trong hệ thống.

Bảng đặc tả chi tiết các Use Case cốt lõi trong hệ thống Secure Chat

Tên Use Case	Điều kiện tiên quyết (Pre-conditions)	Luồng sự kiện chính thành công (Success Scenario)
UC_Auth (Đăng nhập)	Người dùng đã có tài khoản hợp lệ lưu trên MySQL cơ sở dữ liệu.	1. Người dùng nhập Username, Password trên Swing UI.

Cấu hình LLM | Tác nhân Admin thực hiện truy cập quyền Root vào hạ tầng máy chủ Self-hosted. | 1. Admin tiến hành truy cập và chỉnh sửa tệp Modelfile của Ollama. 2. Thực hiện điều chỉnh các tham số kỹ thuật như temperature, top_p hoặc thực hiện chuyển đổi giữa các mô hình Llama 3 và Mistral. 3. Tiến hành khởi động lại container Ollama bằng công cụ Docker-compose CLI.

3.3.1. Đặc tả yêu cầu: Phân hệ Xác thực & Quản lý Phiên làm việc (Kèm sơ đồ Sequence)

Phân tích Yêu cầu Kỹ thuật: Nhóm chức năng Xác thực & Phiên làm việc

Đánh giá An ninh trong Quản lý Phiên và Xác thực: Đối với một nền tảng nhắn tin thực thi mã hóa đầu cuối (E2EE) như Secure Chat, phân hệ Xác thực và Quản lý phiên (Authentication & Session Management) giữ vai trò then chốt. Chức năng này không chỉ giới hạn ở việc kiểm tra tính hợp lệ của thông tin đăng nhập (Tên người dùng/Mật khẩu) mà còn là bước khởi đầu quan trọng để thiết lập ngữ cảnh mật mã học (Cryptographic Context) an toàn ngay tại máy trạm của người dùng.

Đặc tả yêu cầu: Nhóm chức năng Xác thực & Phiên làm việc

Phân tích bài toán an ninh trong Xác thực và Quản lý phiên Trong một hệ thống nhắn tin bảo mật chuẩn mã hóa đầu cuối (E2EE) như Secure Chat, nhóm chức năng Xác thực và Quản lý phiên làm việc (Authentication & Session Management) không chỉ đơn thuần là việc kiểm tra tính đúng đắn của cặp định danh tài khoản và mật khẩu (Username/Password), mà còn là chốt chặn đầu tiên để thiết lập môi trường mật mã học an toàn (Cryptographic Context) tại phía máy trạm (Client-side).

Bài toán đặt ra yêu cầu hệ thống phải giải quyết đồng thời hai thách thức lớn:

Cơ chế xác thực không làm lộ thông tin nhạy cảm: Hệ thống áp dụng mô hình băm mật khẩu nâng cao kết hợp chuỗi muối ngẫu nhiên (Salted Hashing) sử dụng thuật toán PBKDF2 (Password-Based Key Derivation Function 2) với 10,000 vòng lặp tại phía Client, trước khi gửi chuỗi băm (Password Hash) lên Server qua kênh truyền bảo mật HTTPS. Điều này triệt tiêu hoàn toàn nguy cơ rò rỉ mật khẩu thô (Plaintext Password) ngay cả khi đường truyền bị can thiệp bởi các cuộc tấn công MitM.

Quản lý phiên phi trạng thái (Stateless Session Management) dựa trên JWT: Nhằm tối ưu hóa hiệu năng cho hạ tầng container hóa Docker, Core Server không duy trì trạng thái phiên trong bộ nhớ (Memory-session) mà sử dụng cơ chế cấp phát mã thông báo JSON Web Token (JWT). Mã thông báo này chứa thông tin định danh người dùng, thời gian hết hạn (Expiration Time), và

được ký số bằng thuật toán mã hóa đối xứng mã mạnh HMAC-SHA256 với một khóa bí mật (Secret Key) chỉ lưu trữ duy nhất tại môi trường an toàn của Server.

Đặc tả chi tiết các yêu cầu chức năng cốt lõi Nhóm chức năng Xác thực & Phiên làm việc bao gồm 03 chức năng thành phần, được thiết kế nghiêm ngặt để đảm bảo tính an toàn hệ thống:

Đăng ký tài khoản (Sign-up): Tiếp nhận yêu cầu khởi tạo tài khoản mới từ người dùng. Ngoài việc kiểm tra tính duy nhất của trường định danh (Username), Client-side của hệ thống sẽ tự động kích hoạt thư viện `java.security.KeyPairGenerator` để sinh ngẫu nhiên cặp khóa bất đối xứng RSA-2048. Khóa công khai (Public Key) được đẩy lên lưu trữ trong hệ quản trị cơ sở dữ liệu MySQL phục vụ luồng định tuyến mã hóa E2EE sau này. Khóa bí mật (Private Key) được mã hóa AES-256 nội bộ dựa trên Key-derived sinh ra từ mật khẩu của người dùng và ghi bền vững xuống cấu trúc tệp cấu hình trạm cục bộ.

Đăng nhập hệ thống (Sign-in): Xác thực danh tính người dùng thông qua chuỗi băm mật khẩu, thiết lập phiên làm việc thời gian thực (Real-time Session) kết nối trực tiếp đến Socket Server để sẵn sàng nhận và truyền tin nhắn dữ liệu.

Đăng xuất và Thu hồi phiên (Sign-out / Session Revocation): Chấm dứt phiên làm việc hiện tại, gửi lệnh hủy Socket kết nối, xóa thông tin JWT Token trong bộ nhớ tạm (RAM) của Client, đồng thời đưa token hiện tại vào danh sách đen (Blacklist) phía Server để vô hiệu hóa hoàn toàn khả năng tái sử dụng (Replay Attacks).

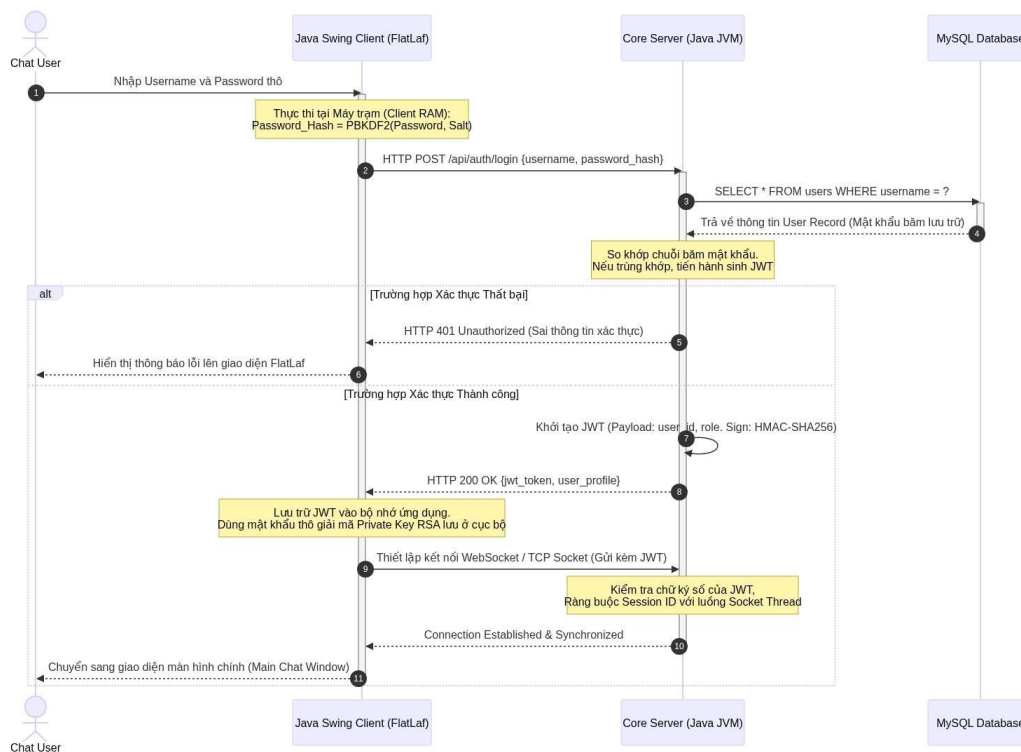
Các tham số nghiệp vụ kỹ thuật chi tiết cùng các điều kiện ràng buộc của nhóm chức năng được hệ thống hóa cụ thể trong bảng đặc tả dưới đây:

Bảng 3.2: Ma trận đặc tả kỹ thuật nhóm chức năng Xác thực và Phiên làm việc

Tên chức năng	Điều kiện tiên quyết	Dữ liệu đầu vào (Payload)	Trạng thái đầu ra / Ràng buộc an ninh
Đăng ký tài khoản	Không	username, password_hash, rsa_public_key	Khởi tạo bản ghi người dùng mới trong MySQL; Trả về trạng thái HTTP 201 Created.
Đăng nhập hệ thống	Tài khoản đã tồn tại hợp lệ trên hệ thống	username, password_hash, client_device_info	Sinh mã JWT Access Token (hạn 24 giờ); Khởi chạy luồng Thread lắng nghe kết nối TCP Socket; Trả về danh bạ.
Đăng xuất hệ thống	Phiên làm việc hiện tại đang hoạt động	jwt_access_token, session_id	Đóng kết nối TCP Socket một cách an toàn; Giải phóng bộ nhớ

Tên chức năng	Điều kiện tiên quyết	Dữ liệu đầu vào (Payload)	Trạng thái đầu ra / Ràng buộc an ninh
			RAM chứa khóa mã hóa cục bộ; Hủy hiệu lực Token trên Server.

Biểu đồ tuần tự (Sequence Diagram) luồng Đăng nhập hệ thống Luồng xử lý kỹ thuật phức tạp của chức năng Đăng nhập hệ thống — từ việc người dùng nhập thông tin trên giao diện FlatLaf cho đến khi thiết lập thành công phiên mã hóa an toàn và đồng bộ kết nối Socket thời gian thực — được mô tả chi tiết qua sơ đồ Sequence Diagram dưới đây:



Hình 18: Biểu đồ tuần tự (Sequence Diagram) luồng xử lý Xác thực và Khởi tạo Phiên làm việc

3.3.2. Đặc tả yêu cầu: Nhóm chức năng Nhắn tin & Trạng thái Real-time

Bản chất kỹ thuật của luồng dữ liệu thời gian thực và Quản lý trạng thái Nhóm chức năng Nhắn tin và cập nhật trạng thái thời gian thực (Real-time Messaging & Presence Management) quyết định trực tiếp đến tính ổn định và độ trễ phản hồi của hệ thống Secure Chat. Bài toán kỹ thuật cốt lõi ở đây là thiết lập một kênh truyền dữ liệu hai chiều có độ trễ cực thấp (Low Latency) giữa Client (Java Swing) và Core Server, đồng thời phải đảm bảo cấu trúc gói tin tuân thủ tuyệt đối giao thức mã hóa đầu cuối (E2EE).

Để giải quyết bài toán phân phối tin nhắn thời gian thực, hệ thống áp dụng giao thức mạng hướng kết nối dựa trên nền tảng **TCP Sockets/WebSockets**. Thay vì sử dụng cơ chế kéo dữ liệu

truyền thống (HTTP Polling) gây lãng phí tài nguyên băng thông và CPU, Core Server duy trì một danh bạ các luồng Thread hoạt động (Active Connection Pool). Khi một Client thiết lập kết nối thành công sau bước xác thực JWT, một luồng Socket Thread riêng biệt sẽ được liên kết với ID của người dùng đó. Sơ đồ này cho phép máy chủ thực hiện cơ chế đẩy dữ liệu chủ động (Server-Push) ngay khi phát hiện có gói tin mã hóa định tuyến đến đích.

Đặc tả chi tiết các yêu cầu chức năng cốt lõi Nhóm chức năng này được chia nhỏ thành các tác vụ nghiệp vụ có tính liên kết chặt chẽ về mặt logic, bao gồm:

Gửi và Định tuyến Tin nhắn E2EE (Send \& Route Message): Tiếp nhận các gói tin đã mã hóa AES-256 từ Client nguồn. Server đọc thông tin header không mã hóa ('recipient_id') để định tuyến dữ liệu thô. Nếu Client đích đang trực tuyến ('ONLINE'), gói tin được đẩy trực tiếp qua Socket. Nếu ngoại tuyến ('OFFLINE'), gói tin được lưu tạm vào hàng đợi tin nhắn chờ (Pending Messages) trong cơ sở dữ liệu MySQL.

Cập nhật trạng thái hiện diện (Presence Tracking): Theo dõi trạng thái hoạt động của người dùng ('ONLINE', 'OFFLINE', 'AWAY'). Khi kết nối Socket bị ngắt đột ngột (do mất mạng hoặc tắt ứng dụng), hệ thống tự động kích hoạt sự kiện phá hủy liên kết ('Disconnect Event'), chuyển đổi trạng thái người dùng trong Database và phát thông báo dạng quảng bá ('Broadcast') tới tất cả các liên hệ có trong danh bạ của người dùng đó.

Xác thực trạng thái tin nhắn (Message Delivery Receipts): Đảm bảo tính minh bạch của luồng tin thông qua 3 trạng thái chuẩn: Đã gửi (Sent), Đã nhận (Delivered), và Đã đọc (Read). Tác vụ này đòi hỏi các gói tin báo hiệu (Signaling Packets) siêu nhẹ được gửi ngược về từ Client nhận để cập nhật trực tiếp lên giao diện người dùng.

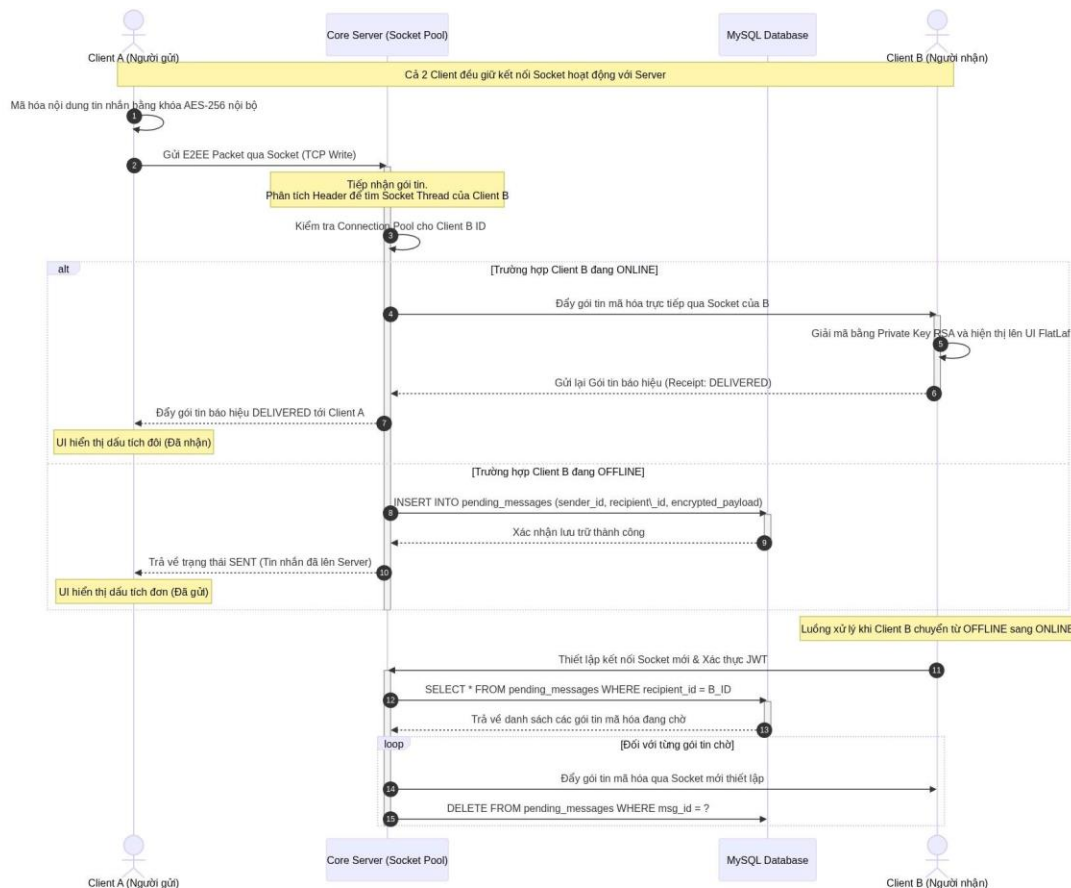
Các ràng buộc kỹ thuật và luồng dữ liệu của nhóm chức năng được tổng hợp chi tiết trong bảng đặc tả dưới đây:

Bảng 3.3: Ma trận đặc tả kỹ thuật nhóm chức năng Nhắn tin & Trạng thái Real-time

Tên chức năng	Điều kiện tiên quyết	Dữ liệu đầu vào (Payload)	Trạng thái đầu ra / Ràng buộc an ninh
Gửi tin nhắn E2EE	Phiên Socket đang hoạt động (ONLINE)	recipient_id, encrypted_msg, encrypted_key, crypto_iv, auth_tag	Định tuyến thời gian thực đến Client đích qua Socket Thread hoặc lưu vào MySQL nếu User offline. Trả về mã định danh tin nhắn (msg_id).
Đồng bộ trạng thái hiện diện	Trạng thái kết nối mạng của Client thay đổi	user_id, presence_status (ONLINE/OFFLINE)	Phát gói tin cập nhật trạng thái tới tất cả các Client là bạn bè đang kết nối. Cập nhật

Tên chức năng	Điều kiện tiên quyết	Dữ liệu đầu vào (Payload)	Trạng thái đầu ra / Ràng buộc an ninh
Báo cáo trạng thái tin nhắn	Client nhận tiếp nhận hoặc mở đọc gói tin	msg_id, sender_id, receipt_type (DELIVERED/ READ)	Trạng thái đầu ra / Ràng buộc an ninh trường last_seen trong DB. Gửi gói tin báo hiệu trực tiếp cho người gửi gốc để cập nhật icon trạng thái trên bong bóng chat FlatLaf.

Quy trình kỹ thuật và Luồng xử lý tin nhắn thời gian thực Sự kết hợp giữa luồng Socket Thread, cơ chế kiểm tra trạng thái tại cơ sở dữ liệu và quá trình cập nhật bất đồng bộ lên giao diện người dùng được mô tả chi tiết thông qua sơ đồ Sequence Diagram dưới đây:



Hình 19: Biểu đồ tuần tự (Sequence Diagram) luồng định tuyến tin nhắn E2EE và xử lý trạng thái đồng bộ thời gian thực

3.3.3. Đặc tả yêu cầu: Nhóm chức năng Giao tiếp đa phương tiện (Video call)

Nhóm chức năng Giao tiếp đa phương tiện (Video/Audio Call) là một phân hệ cốt lõi trong hệ thống Secure Chat, cung cấp giải pháp hội nghị truyền hình thời gian thực độ trễ thấp trực tiếp trên nền tảng Java Swing Client. Phân hệ này được đặc tả dựa trên kiến trúc phân tán Jitsi Meet Stack (Prosody, Jicofo, JVB) tự vận hành (Self-hosted) nhằm tối ưu hóa băng thông, đồng thời tích hợp các giao thức bảo mật lớp truyền tải để bảo vệ luồng dữ liệu truyền thông trước các nguy cơ nghe lén.

1. Phân tích yêu cầu chức năng và Phi chức năng chuyên sâu

Để hiện thực hóa một hệ thống truyền thông đa phương tiện đạt chuẩn học thuật và ứng dụng thực tiễn, các yêu cầu hệ thống được phân rõ cụ thể như sau:

Yêu cầu chức năng (Functional Requirements):

Khởi tạo và Điều phối phòng họp (Conference Orchestration): Cho phép người dùng tạo phòng gọi mới hoặc tham gia vào một phòng đang hoạt động dựa trên mã phòng định danh duy nhất (Room UUID).

Xử lý luồng Media cục bộ (Local Media Handling): Cấp quyền và bắt trọn dòng dữ liệu từ thiết bị phần cứng (Camera, Microphone) của máy trạm, chuyển đổi định dạng và nạp vào bộ mã hóa WebRTC.

Điều khiển trạng thái thiết bị (Device Mutability): Cho phép bật/tắt (Toggle) Camera và Microphone theo thời gian thực; trạng thái thay đổi phải được đồng bộ tức thì đến tất cả các Client khác trong phòng thông qua kênh báo hiệu XMPP.

Yêu cầu phi chức năng (Non-functional Requirements):

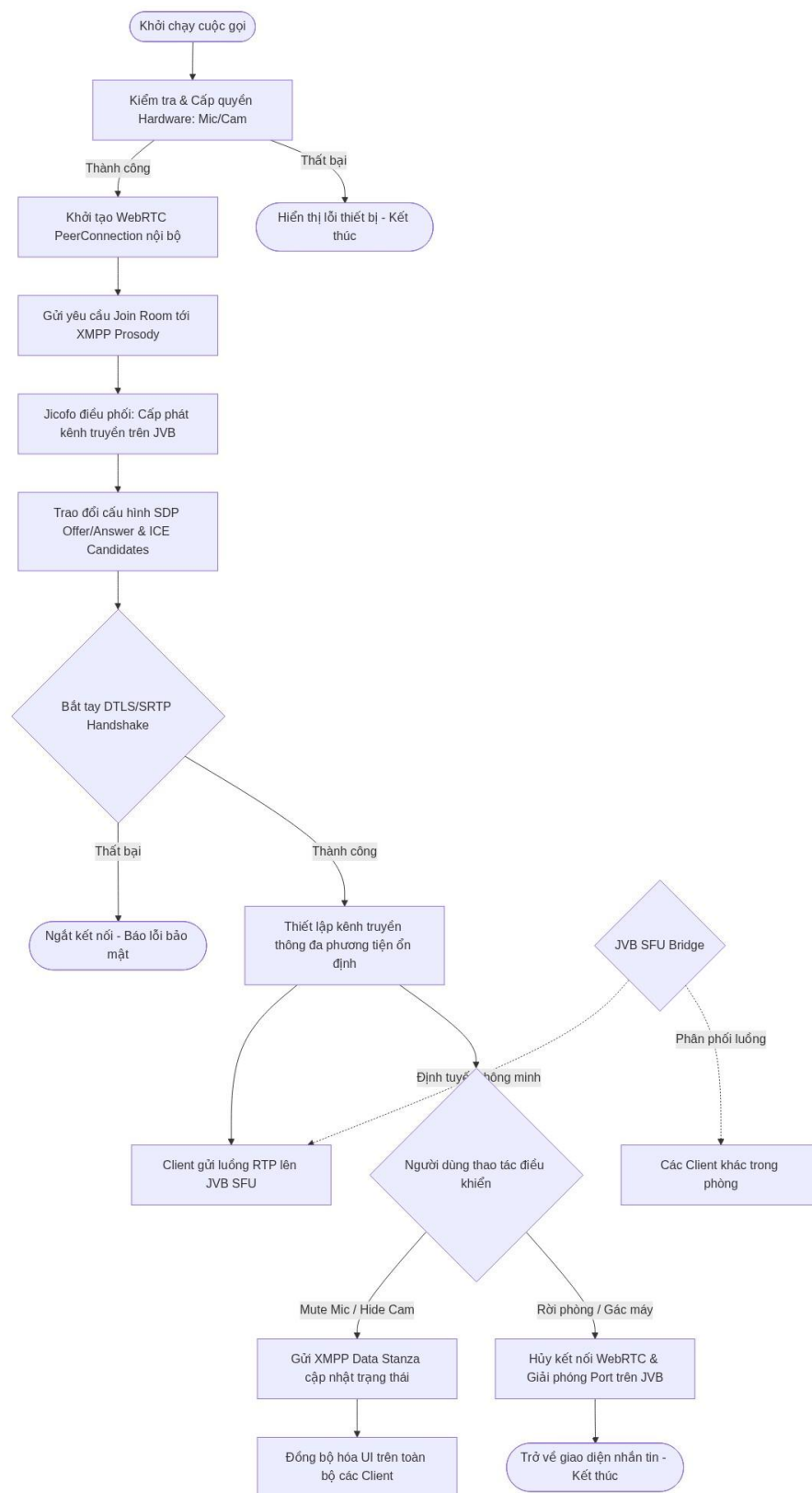
Độ trễ truyền tải (End-to-End Latency): Tổng độ trễ truyền phát âm thanh và hình ảnh giữa các Client thông qua SFU Jitsi Videobridge (JVB) phải duy trì dưới mức \$200\ extms\$ trong điều kiện mạng tiêu chuẩn.

Độ ổn định luồng (Jitter và Packet Loss Resilience): Hệ thống phải có khả năng bù mất gói (Packet Loss Concealment - PLC) và duy trì khung hình mượt mà ngay cả khi tỷ lệ mất gói tin đạt ngưỡng \$15\%\$.

Bảo mật lớp truyền tải (Transport Security): Toàn bộ các luồng dữ liệu đa phương tiện RTP (Real-time Transport Protocol) bắt buộc phải được mã hóa bằng giao thức SRTP (Secure Real-time Transport Protocol) kết hợp với cơ chế trao đổi khóa mã hóa DTLS (Datagram Transport Layer Security).

2. Biểu đồ luồng dữ liệu logic và Trạng thái kết nối cuộc gọi

Quy trình vận hành của nhóm chức năng Video Call chuyển dịch qua nhiều trạng thái logic phức tạp, từ pha thu thập tín hiệu phần cứng, thiết lập phiên thông qua Signaling Server (Prosody và Jicofo), cho đến pha truyền thông trực tiếp qua SFU Bridge. Luồng kiến trúc kỹ thuật này được mô tả chi tiết qua sơ đồ dưới đây:



Hình 20: Biểu đồ luồng logic tiền xử lý và định tuyến luồng Video Call qua hạ tầng Jitsi.

3. Kiến trúc kỹ thuật và Cơ chế mã hóa bảo mật truyền thông (DTLS-SRTP)

Điểm mấu chốt mang tính học thuật trong đặc tả chức năng Video Call của Secure Chat là việc thực thi kiến trúc bảo mật phân tầng tại máy trạm Client Java Swing kết hợp Jitsi Stack. Hệ thống áp dụng cơ chế thiết lập khóa mã hóa động **DTLS-SRTP** nhằm triệt tiêu nguy cơ tấn công xen giữa (Man-in-the-middle attack).

Cơ chế thiết lập khóa (DTLS Handshake): Giao thức WebRTC chạy trên ứng dụng Client sử dụng DTLS (phiên bản 1.2) dựa trên giao thức UDP để thực hiện quá trình bắt tay bảo mật với JVB. Trong pha bắt tay này, hai đầu nút trao đổi chứng chỉ số tự ký (Self-signed certificates) kèm theo mã băm chứng chỉ (Certificate fingerprints) thông qua kênh báo hiệu bảo mật XMPP. Quá trình này giúp xác thực chính xác danh tính thực thể kết nối mà không cần đến hạ tầng khóa công khai (PKI) phức tạp.

Mã hóa dòng dữ liệu (SRTP Media Encryption): Sau khi pha bắt tay DTLS hoàn tất thành công, hai bên sẽ trích xuất ra một chuỗi byte ngẫu nhiên entropy cao từ tiến trình DTLS để làm khóa master (Master Key) cho luồng mã hóa đối xứng đối với dữ liệu truyền thông. Luồng RTP chứa dữ liệu âm thanh (mã hóa bằng codec Opus) và dữ liệu hình ảnh (mã hóa bằng codec VP8 hoặc H.264) sẽ lập tức được chuyển đổi sang định dạng mã hóa SRTP bằng thuật toán AES-128-ICM (Integer Counter Mode). Cơ chế này đảm bảo dữ liệu truyền tải qua môi trường Internet hoàn toàn là các khối dữ liệu nhiễu sai lệch, ngăn chặn tuyệt đối việc đánh cắp luồng đa phương tiện tại các nút trung gian.

4. Bảng đặc tả yêu cầu chi tiết của các Use Case thành phần

Để cung cấp tài liệu kỹ thuật phục vụ giai đoạn phát triển và kiểm thử hệ thống (Testing phase), Bảng 1 mô tả cấu trúc đặc tả yêu cầu chi tiết cho hai Use Case cốt lõi thuộc phân hệ Giao tiếp đa phương tiện.

Bảng đặc tả yêu cầu chức năng phân hệ Video Call

Trường thông tin	Nội dung đặc tả chi tiết
Mã chức năng: UC_MEDIA_01 - Khởi tạo cuộc gọi Video Call	
Tác nhân (Actors)	Người dùng ứng dụng Client (User)
Mục tiêu (Goal)	Thiết lập phòng gọi mới và truyền phát luồng dữ liệu đa phương tiện lên máy chủ JVB SFU.
Tiền điều kiện	1. Người dùng đã xác thực hệ thống thành công.
	2. Thiết bị phần cứng (Camera, Microphone) hoạt động bình thường và đã được cấp quyền truy cập.

Luồng sự kiện chính	1. Người dùng nhấn nút biểu tượng “Video Call” trên giao diện hội thoại Java Swing. 2. Hệ thống Client sinh mã Room UUID ngẫu nhiên và gửi yêu cầu tạo phòng chat đa người dùng (MUC) tới Prosody. 3. Jicofo tiếp nhận tín hiệu, khởi tạo phiên Colibri kiểm tra và cấp phát cổng UDP (Port từ dải 10000-20000) trên JVB. 4. Client nhận cấu hình SDP Offer từ Jicofo, thực hiện bắt tay cấu hình mạng (ICE/STUN) và thiết lập kết nối WebRTC. 5. Kích hoạt luồng mã hóa DTLS-SRTP, Client bắt đầu đẩy luồng dữ liệu hình ảnh/âm thanh cục bộ lên Server.
Hậu điều kiện	Phòng gọi hoạt động ổn định trên JVB, sẵn sàng tiếp nhận các thành viên khác tham gia.

Mã chức năng: UC_MEDIA_02 - Đồng bộ hóa trạng thái thiết bị (Mute/Unmute)

Mã chức năng	UC_MEDIA_02
Tên chức năng	Đồng bộ hóa trạng thái thiết bị (Mute/Unmute)
Tác nhân (Actors)	Người dùng đang tham gia phòng gọi (Participant)
Mục tiêu (Goal)	Thay đổi trạng thái hoạt động của Camera/Microphone cá nhân và đồng bộ lên giao diện của toàn phòng.

Luồng sự kiện chính	1. Người dùng nhấn nút “Mute Microphone” trên thanh công cụ FlatLaf UI. 2. Hệ thống Client lập tức ngắt luồng thu âm từ lớp TargetDataLine phân cứng hoặc gán trạng thái vô hiệu hóa (Disabled) cho MediaStreamTrack của WebRTC. 3. Client đóng gói gói tin trạng thái dạng XMPP Presence Stanza chứa cờ hiệu true gửi tới Prosody. 4. Prosody khuếch tán (Broadcast) gói tin này tới tất cả các Client khác hiện diện trong phòng. 5. Các Client nhận tin phân tách gói dữ liệu và cập nhật biểu tượng “Tắt tiếng” trên avatar của người dùng tương ứng thông qua lệnh SwingUtilities.invokeLater().
Hậu điều kiện	Âm thanh từ Client thao tác bị ngắt hoàn toàn; giao diện toàn hệ thống hiển thị chính xác trạng thái thực tế.

3.3.4. Đặc tả yêu cầu: Nhóm chức năng Truyền tải dữ liệu an toàn

Phân tích bài toán an ninh và Giải pháp kỹ thuật trong chuyển tệp tin Trong các hệ thống cộng tác doanh nghiệp, việc chia sẻ tài liệu, mã nguồn và dữ liệu đa phương tiện chiếm tỷ trọng tài nguyên lớn và là đối tượng mục tiêu hàng đầu của các cuộc tấn công khai thác dữ liệu. Khác với tin nhắn văn bản có kích thước nhỏ gọn (vài byte đến vài kilobyte) có thể xử lý hoàn toàn trong bộ nhớ RAM, tệp tin (Files/Attachments) thường có kích thước từ vài megabyte đến hàng gigabyte. Do đó, việc áp dụng trực tiếp thuật toán mã hóa bất đối xứng RSA-2048 lên toàn bộ nội dung tệp tin là bất khả thi về mặt hiệu năng tính toán ($O(N^3)$).

Để giải quyết triệt để bài toán này, hệ thống Secure Chat thiết kế một giao thức truyền tải tệp tin an toàn dựa trên mô hình mã hóa lai kết hợp cơ chế băm kiểm tra tính toàn vẹn (Data Integrity Check). Điểm cốt lõi của giải pháp là việc sinh ngẫu nhiên một khóa đối xứng dùng một lần (*Ephemeral Symmetric Key*) gọi là FileTransferKey cho mỗi phiên truyền tệp. Tệp tin thô được mã hóa cục bộ thành một khối dữ liệu nhị phân bất định (Encrypted Blob) trước khi truyền qua Internet. Phía Server đóng vai trò là một kho chứa đối tượng mù (Blind Object Storage), lưu trữ tệp tin dưới dạng đã mã hóa và hoàn toàn không có khả năng nhận biết cấu trúc hay nội dung thực tế của tệp tin.

Đặc tả chi tiết các yêu cầu chức năng cốt lõi Nhóm chức năng Truyền tải dữ liệu an toàn bao gồm 03 tác vụ nghiệp vụ có sự liên kết chặt chẽ từ phía Client gửi, Server trung gian đến Client nhận:

Mã hóa và Tải lên tệp tin (Encrypt & Upload File): Khi người dùng thực hiện hành vi gửi tệp, Client sử dụng bộ sinh số ngẫu nhiên an toàn mật mã (SecureRandom) để tạo FileTransferKey (256-bit) và một Vector khởi tạo (IV - 96-bit). Tệp tin được mã hóa bằng thuật toán AES-256-

GCM. Song song với đó, hệ thống tính toán giá trị băm mã hóa SHA-256 của tệp tin gốc để làm giá trị đối chiếu tính toàn vẹn (Checksum). Khối mật mã (Encrypted Blob) được tải lên Server thông qua giao thức HTTP POST Multi-part Request.

Phân phối và Trao đổi khóa bảo mật (Key Distribution): Để người nhận có thể giải mã tệp tin, khóa `FileTransferKey`, `IV` và chuỗi `Checksum` được đóng gói thành một cấu trúc siêu dữ liệu (File Metadata). Gói Metadata này được mã hóa bằng khóa công khai RSA-2048 của người nhận và định tuyến qua kênh chat E2EE thời gian thực (được cấu hình tại mục 3.3.2).

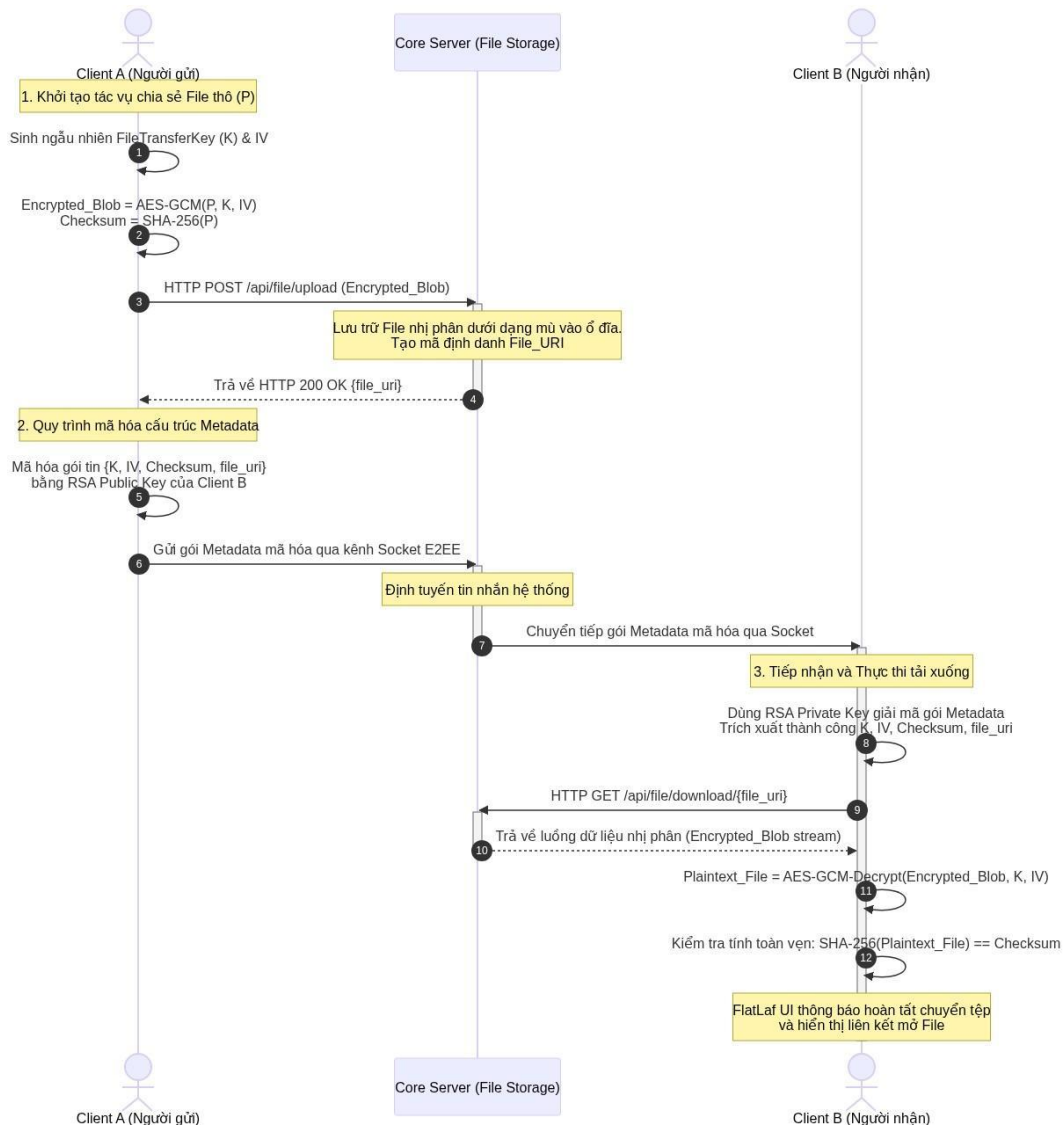
Tải xuống và Giải mã tệp tin (Download & Decrypt File): Client nhận sau khi nhận được gói Metadata mã hóa sẽ dùng khóa Private Key RSA của mình để giải mã, thu hồi lại `FileTransferKey`, `IV` và `Checksum`. Kế tiếp, ứng dụng gửi yêu cầu HTTP GET lên Server để tải Encrypted Blob về bộ nhớ đệm, thực hiện giải mã bằng AES-256-GCM và đối chiếu lại chuỗi SHA-256 để đảm bảo tệp tin không bị lỗi hoặc bị can thiệp trong quá trình truyền tải.

Các tham số nghiệp vụ cấu trúc và ràng buộc hệ thống được chi tiết hóa trong bảng đặc tả kỹ thuật dưới đây:

Bảng 3.4: Ma trận đặc tả kỹ thuật nhóm chức năng Truyền tải dữ liệu an toàn

Tên chức năng	Điều kiện tiên quyết	Dữ liệu đầu vào (Payload)	Trạng thái đầu ra / Ràng buộc an ninh
Tải lên tệp an toàn	Kích thước tệp không vượt quá giới hạn cấu hình (VD: 100MB)	file_binary_blob (đã mã hóa AES), recipient_id	Lưu trữ tệp vào thư mục cô lập trên Server Docker. Trả về mã băm định danh tệp (file_uri).
Trao đổi File Metadata	Tải lên tệp thành công và nhận được file_uri	file_uri, encrypted_transfer_key, file_iv, sha256_checksum	Gói tin Metadata được truyền dưới dạng tin nhắn E2EE đến Client nhận qua giao thức Socket.
Tải xuống và Giải mã	Nhận được gói Metadata và có khóa Private Key tương ứng	file_uri, decrypted_transfer_key, file_iv	Tệp tin được khôi phục nguyên bản về thư mục Downloads của máy trạm; Hiện thị thanh tiến trình (Progress Bar).

Luồng xử lý kỹ thuật tổng thể của quy trình truyền tải dữ liệu Quy trình thực thi tuần tự của nhóm chức năng này yêu cầu sự phối hợp bất đồng bộ giữa luồng tải dữ liệu nặng (REST API HTTP) và luồng báo hiệu cấu trúc (Socket) để tối ưu hóa hiệu năng mạng. Toàn bộ luồng kỹ thuật được mô tả chi tiết qua sơ đồ Sequence Diagram dưới đây:



Hình 21: Biểu đồ tuần tự (Sequence Diagram) luồng xử lý mã hóa lai trong truyền tải dữ liệu an toàn

3.3.5. Đặc tả yêu cầu: Nhóm chức năng Tương tác AI (Ollama Chatbot)

Phân hệ Tương tác AI (Ollama Chatbot) cung cấp giải pháp trợ lý ảo thông minh hoạt động hoàn toàn cục bộ (Local AI) trên hạ tầng tự vận hành (Self-hosted Docker) của hệ thống Secure Chat. Mục tiêu học thuật và kỹ thuật tối cao của phân hệ này là tích hợp năng lực xử lý ngôn ngữ tự nhiên từ các mô hình lớn (Llama 3 / Mistral) mà không làm rò rỉ bất kỳ dữ liệu hội thoại nào của người dùng ra môi trường Internet. Phân hệ được đặc tả chi tiết từ kiến trúc điều phối bất đồng bộ đến cơ chế tiền xử lý ngữ cảnh an toàn tại Client Java Swing.

1. Phân tích yêu cầu chức năng và Phi chức năng chuyên sâu

Yêu cầu chức năng (Functional Requirements):

Xử lý hội thoại thời gian thực (Contextual Dialogue): Tiếp nhận câu hỏi (Prompt) từ người dùng, tự động đóng gói ngữ cảnh lịch sử trò chuyện liên kết và chuyển tiếp tới mô hình ngôn ngữ lớn (LLM).

Phản hồi luồng văn bản (Stream Response Processing): Tiếp nhận các phần tử từ (Tokens) sinh ra từ Ollama theo cơ chế Server-Sent Events (SSE) và kết xuất trực tiếp lên giao diện đồ họa thông qua cơ chế không đồng bộ.

Quản lý cửa sổ ngữ cảnh cục bộ (Context Window Tracking): Tự động tính toán lượng Token đầu vào, thực hiện cắt tỉa (Pruning) lịch sử chat dựa trên giải thuật cửa sổ trượt nhằm tối ưu hóa dung lượng bộ nhớ đệm (VRAM/RAM) của máy chủ.

Yêu cầu phi chức năng (Non-functional Requirements):

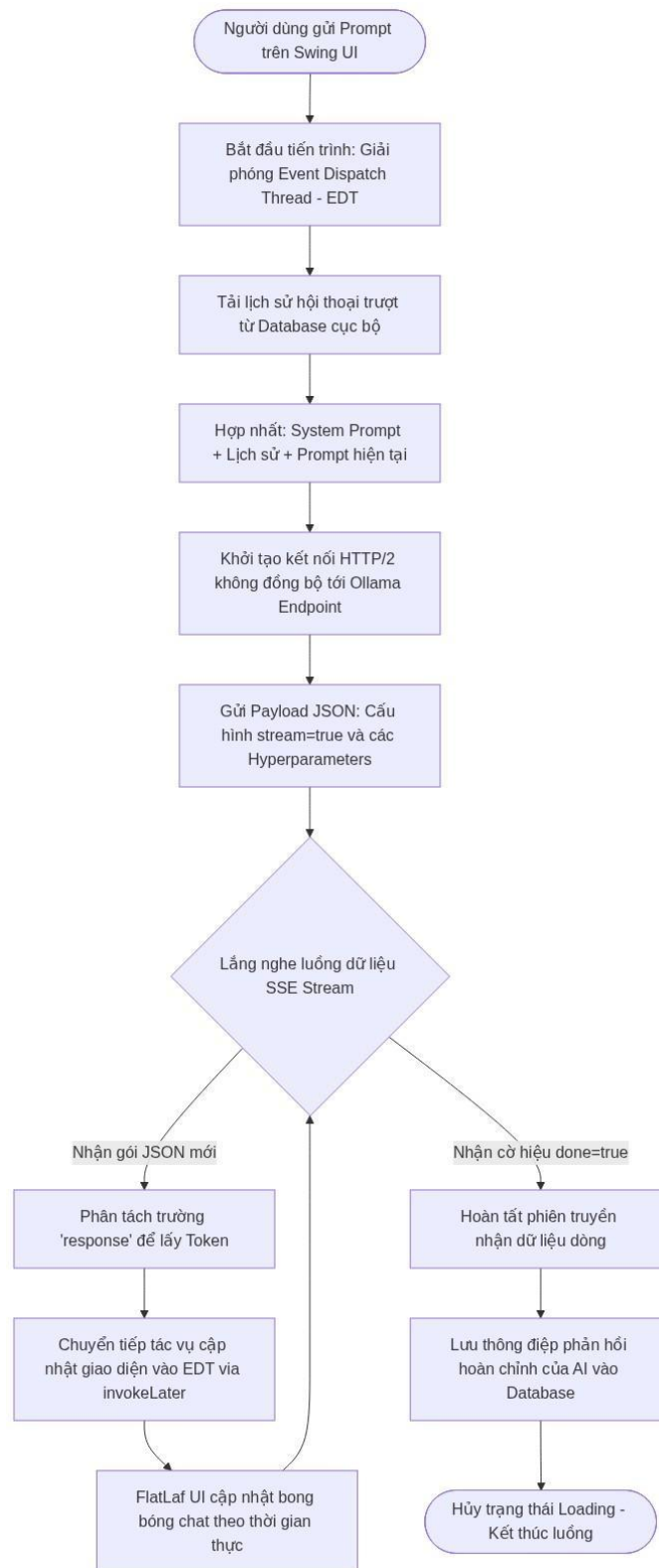
Bảo mật dữ liệu tuyệt đối (Zero Cloud Leakage): Toàn bộ quy trình từ gửi câu hỏi đến suy luận dữ liệu bắt buộc phải thực thi trong mạng nội bộ của hạ tầng Docker. Nghiêm cấm việc lưu trữ tạm thời (Caching) hoặc chuyển tiếp cấu trúc văn bản rõ lên bất kỳ API đám mây nào.

Thời gian đáp ứng dòng đầu tiên (Time-to-First-Token - TTFT): Thời gian từ khi người dùng nhấn gửi đến khi Token đầu tiên xuất hiện trên bong bóng chat của giao diện FlatLaf phải nhỏ hơn \$1.5 \times 10^3\$ đối với cấu hình máy chủ tiêu chuẩn.

Định dạng kết xuất đồ họa (UI Rendering Resilience): Thành phần hiển thị phải phân tách và render chính xác mã nguồn Markdown, cú pháp LaTeX toán học, và cấu trúc bảng biểu do LLM trả về mà không làm phá vỡ kiến trúc layout đồ họa của Java Swing.

2. Biểu đồ luồng trạng thái xử lý Prompt và Phân tách dữ liệu

Quy trình kỹ thuật xử lý yêu cầu tương tác AI bao gồm các công đoạn tiền xử lý chuỗi ký tự, quản lý tiến trình (Threading) và phân rã các gói tin JSON trả về theo dạng dòng được đặc tả trực quan qua sơ đồ dưới đây:



Hình 22: Luồng logic xử lý bất đồng bộ cấu trúc Prompt và hiển thị dữ liệu dạng dòng từ Ollama API.

3. Kiến trúc tiền xử lý ngữ cảnh an toàn và Kỹ thuật điều phối luồng (Multi-threading)

Một rủi ro lớn trong các ứng dụng tích hợp chatbot thông thường là hiện tượng đóng băng giao diện khi mô hình đang thực hiện các tác vụ tính toán nặng (Computationally intensive tasks). Hệ thống Secure Chat giải quyết triệt để bài toán này thông qua cấu trúc phân tách luồng nghiêm ngặt:

Cơ chế Thread Pool điều phối tác vụ (Worker Threads): Khi sự kiện gửi tin nhắn được kích hoạt, luồng EDT chính của Swing ngay lập tức ủy thác toàn bộ logic kết nối mạng và tiền xử lý văn bản cho một tiểu trình con nằm trong cấu trúc 'ExecutorService' (Fixed Thread Pool). EDT quay trở lại trạng thái lắng nghe hành vi của người dùng trên giao diện, loại bỏ hoàn toàn tình trạng treo ứng dụng.

Giải thuật cửa sổ trượt quản lý ngữ cảnh (Sliding Window Algorithm): Để ngăn ngừa hiện tượng vượt ngưỡng bộ nhớ của mô hình khi hội thoại kéo dài, hệ thống cài đặt bộ lọc trượt tại tầng Client. Trước khi gửi yêu cầu tới Ollama, một thuật toán tính toán độ dài chuỗi ký tự được áp dụng để ước lượng tổng số lượng Token dự kiến theo công thức xấp xỉ
Nếu tổng số lượng Token vượt quá dung lượng bộ nhớ được cấu hình (ví dụ: hệ thống sẽ tiến hành loại bỏ tuần tự các cặp tin nhắn cũ nhất trong bộ đệm, chỉ giữ lại các thông điệp cốt lõi gần nhất kèm theo cấu trúc System Prompt cố định để gửi đi.

4. Bảng đặc tả yêu cầu chi tiết của các Use Case phân hệ AI

Bảng 1 cung cấp tài liệu đặc tả cấu trúc chi tiết cho các điều kiện biên, kịch bản chính và luồng xử lý ngoại lệ của hai Use Case cốt lõi thuộc phân hệ Tương tác AI.

Bảng đặc tả yêu cầu chức năng phân hệ Ollama Chatbot

Trường thông tin	Nội dung đặc tả chi tiết
Mã chức năng: UC_AI_01 - Gửi câu hỏi và Nhận phản hồi dạng Stream	
Tác nhân (Actors)	Người dùng ứng dụng Client (User)
Mục tiêu (Goal)	Nhận câu trả lời từ mô hình LLM nội bộ hiển thị liên tục dưới dạng từ khóa xuất hiện dần trên khung chat.
Tiền điều kiện	Container Ollama chứa mô hình Llama 3 / Mistral đang hoạt động bình thường trong mạng nội bộ Docker.

Luồng sự kiện chính (Tiếp tục)	1. Người dùng nhập nội dung yêu cầu vào khung hội thoại với Trợ lý AI và nhấn nút gửi. 2. Hệ thống chuyển đổi giao diện sang trạng thái chờ, khởi tạo một Worker Thread riêng biệt để xử lý. 3. Hệ thống Client đóng gói cấu trúc dữ liệu gồm: System Prompt định hình vai trò, mảng chứa ngữ cảnh lịch sử đối thoại trượt, và nội dung câu hỏi mới. 4. Thực hiện lệnh HTTP POST không đồng bộ tới địa chỉ <code>http://localhost:11434/api/chat</code> với tham số cấu hình <code>"stream": true</code>. 5. Khi nhận được từng khối dữ liệu (Data chunks) chứa cấu trúc JSON từ luồng mạng, hệ thống phân tách chuỗi Token và sử dụng <code>SwingUtilities.invokeLater()</code> để cập nhật text liên tục vào bong bóng chat.
Luồng ngoại lệ	Trường hợp lỗi kết nối mạng máy chủ AI (Mất kết nối Container): Nếu quá trình kết nối ném ra ngoại lệ <code>ConnectException</code> hoặc mã trạng thái HTTP trả về khác 200, Worker Thread sẽ phát tín hiệu lỗi về EDT. Giao diện Swing dừng hiệu ứng loading, hiển thị bong bóng chat màu đỏ với nội dung thông báo lỗi: <i>“Không thể kết nối đến máy chủ AI cục bộ. Vui lòng kiểm tra lại hạ tầng Docker.”</i>
Hậu điều kiện	Toàn bộ câu trả lời hoàn chỉnh được kết xuất thành công; nội dung phản hồi được lưu vết vào cơ sở dữ liệu MySQL cục bộ phục vụ cho các phiên làm việc tiếp theo.

3.4. Thiết kế Hướng đối tượng (Class Diagram cho các module View, Model, Controller)

Thiết kế hướng đối tượng của hệ thống Secure Chat tuân thủ chặt chẽ mô hình kiến trúc MVC (Model-View-Controller). Việc tách biệt các tầng thành phần giúp hệ thống cô lập được logic giao diện đồ họa (Java Swing / FlatLaf) khỏi các tiến trình xử lý nghiệp vụ phức tạp như mã hóa đầu cuối (E2EE) và giao tiếp mạng không đồng bộ với Jitsi Stack hay Ollama API. Sự phân tầng này tăng cường khả năng mở rộng, đơn giản hóa quy trình bảo trì và kiểm thử độc lập (Unit Testing) từng thành phần hệ thống.

1. Kiến trúc phân tầng cấu trúc cấu thành hệ thống

Mỗi thành phần trong sơ đồ lớp đại diện cho một tầng chức năng riêng biệt được thiết kế theo các nguyên lý cấu trúc phần cứng và thiết kế hướng đối tượng nâng cao:

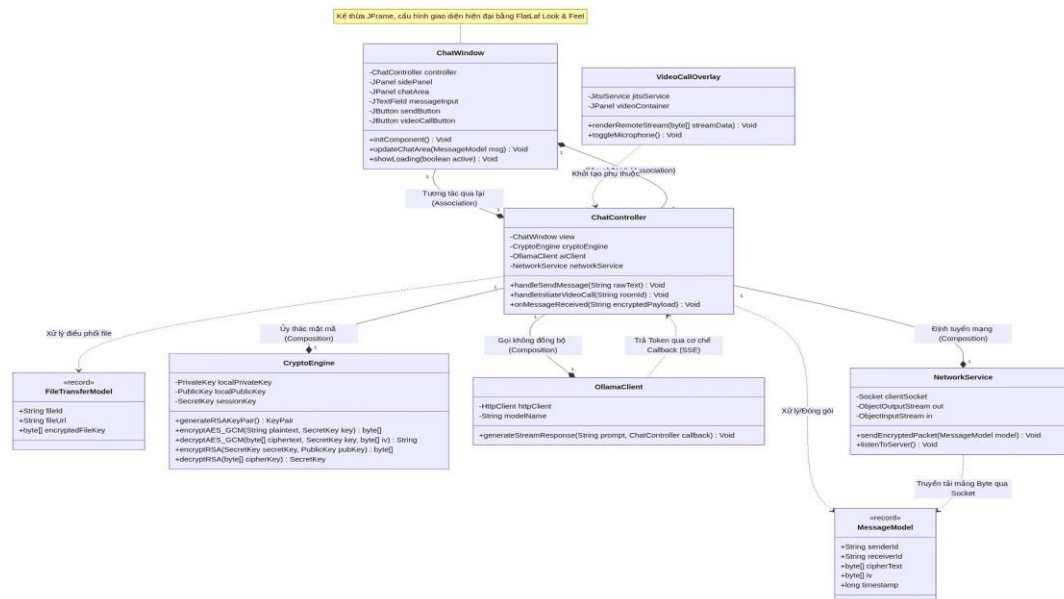
Tầng View (Giao diện đồ họa): Kế thừa từ các thành phần nền tảng của Java Swing ('JFrame', 'JPanel') và được cấu hình giao diện thông qua thư viện 'FlatLaf'. Các lớp View chịu trách nhiệm kết xuất đồ họa (hiệu ứng Glassmorphic, Chat Bubbles dạng Gradient) và thu thập sự kiện tương tác từ người dùng, tuyệt đối không chứa logic nghiệp vụ mật mã hay định tuyến dữ liệu.

Tầng Controller (Điều khiển logic nghiệp vụ): Đóng vai trò là trung tâm điều phối và quản lý luồng dữ liệu (Orchestrator). Controller lắng nghe sự kiện từ View, kích hoạt các tiến trình bất đồng bộ thông qua các Worker Thread, tương tác với tầng Model hoặc gọi API bên ngoài (Ollama/Jitsi), sau đó cập nhật kết quả lại giao diện thông qua cơ chế 'SwingUtilities.invokeLater()'.

Tầng Model & Service (Mô hình dữ liệu và Dịch vụ nền): Chứa các thực thể dữ liệu bất biến (Java Records) như tin nhắn, thông tin tệp tin, cấu hình khóa công khai. Đi kèm là các lớp xử lý hạ tầng đóng gói chuyên sâu ('CryptoEngine', 'NetworkSocket', 'OllamaClient'), quản lý trạng thái phiên làm việc và thực thi các giải thuật tính toán số học lớn (RSA, AES-GCM).

2. Biểu đồ lớp hệ thống (System Class Diagram)

Sơ đồ cấu trúc lớp dưới đây mô tả chi tiết các thuộc tính (Attributes), phương thức (Methods) cùng mối quan hệ phụ thuộc (Dependency), kế thừa (Inheritance), và kết hợp (Association) giữa ba thành phần View, Model, Controller trong module lõi của hệ thống Secure Chat:



Hình 23: Sơ đồ lớp hướng đối tượng tổng thể mô tả cấu trúc kiến trúc MVC của hệ thống.

3. Phân tích cơ chế tương tác và Các mẫu thiết kế (Design Patterns)

Để tối ưu hóa sự vận hành của kiến trúc phân lớp nêu trên, hệ thống Secure Chat tích hợp các mẫu thiết kế hướng đối tượng kinh điển nhằm giải quyết các thách thức kỹ thuật chuyên sâu:

Mẫu thiết kế Người quan sát (Observer Pattern): Được triển khai giữa lớp `NetworkService` và `ChatController`. Lớp `NetworkService` duy trì một luồng chạy ngầm liên tục lắng nghe dữ liệu từ Socket. Khi nhận được một gói tin `MessageModel` đã mã hóa từ Server, `NetworkService` đóng vai trò là Subject sẽ phát đi tín hiệu sự kiện, tự động gọi phương thức callback `onMessageReceived()` của `ChatController` để tiến hành giải mã mà không cần Controller phải liên tục thăm dò trạng thái mạng (Polling).

Mẫu thiết kế Mặt tiền (Facade Pattern): Lớp `CryptoEngine` đóng vai trò là một Facade, che giấu toàn bộ sự phức tạp của kiến trúc mật mã Java Cryptography Extension (JCE). Khớp nối các tiến trình thiết lập `Cipher.getInstance()`, khởi tạo `GCMParameterSpec`, bấm kéo giãn khóa `PBKDF2` được đóng gói gọn trong các hàm tường minh như `encryptAES_GCM()` hay `decryptRSA()`. `ChatController` chỉ cần gọi hàm ở mức trừu tượng cao, giúp giảm thiểu tối đa sai sót lập trình gây lỗ hổng bảo mật rò rỉ khóa.

Cơ chế giải phóng Thread thông qua Callback: Khi `ChatController` chuyển giao một Prompt văn bản cho `OllamaClient`, nó đồng thời truyền chính bản thân nó (`this`) làm tham số Callback. `OllamaClient` sử dụng một luồng Worker riêng biệt để thực hiện kết nối HTTP/2. Mỗi khi nhận được Token từ dòng dữ liệu Server-Sent Events, nó gọi ngược lại phương thức cập nhật của `ChatController`, kích hoạt tiến trình nạp phần tử văn bản an toàn vào giao diện mà không phá vỡ nguyên lý đơn luồng (Single-thread rule) của Java Swing.

4. Đặc tả giao diện lập trình các lớp lõi (Class Interface Specifications)

Bảng 1 cung cấp chi tiết các đặc tả phương thức, kiểu dữ liệu đầu vào và đầu ra của các lớp chịu trách nhiệm xử lý logic nghiệp vụ và mã hóa mật mã trong hệ thống.

Đặc tả phương thức kỹ thuật của các lớp dịch vụ cốt lõi

Tên Lớp / Interface	Chữ ký Phương thức (Method Signature)	Mô tả chức năng học thuật và Ràng buộc dữ liệu
CryptoEngine	<code>encryptAES_GCM(String plaintext, SecretKey key) : byte[]</code>	Mã hóa thông điệp bằng AES-256 chế độ GCM. Tự động sinh IV ngẫu nhiên <i>12extbytes</i> và gộp vào đầu mảng byte kết quả.
	<code>decryptRSA(byte[] cipherKey) : SecretKey</code>	Sử dụng Private Key RSA-2048 lưu trong bộ nhớ máy để giải mã Session Key của phiên trò chuyện.

Tên Lớp / Interface	Chữ ký Phương thức (Method Signature)	Mô tả chức năng học thuật và Ràng buộc dữ liệu
ChatController	handleSendMessage(String rawText) : void	Đọc văn bản rõ từ ChatWindow, gọi CryptoEngine mã hóa thành MessageModel, chuyển sang NetworkService gửi qua mạng.
	onMessageReceived(String encryptedPayload) : void	Nhận bản tin dạng chuỗi mã hóa, bóc tách cấu trúc, giải mã và đẩy text sạch vào giao diện qua luồng EDT.
OllamaClient	generateStreamResponse(String prompt, ChatController cb) : void	Khởi tạo kết nối HTTP/2 SSE tới Container Ollama, truyền nhận mảng token và gọi hàm cb.updateChatArea() theo dạng dòng.

3.5. Thiết kế Dữ liệu (Database Schema cho Users, Messages - Kèm bảng mô tả các trường DB)

Trong một hệ thống nhắn tin bảo mật tuân thủ nghiêm ngặt nguyên tắc Mã hóa đầu cuối (E2EE) và vận hành tự chủ (Self-hosted) như Secure Chat, kiến trúc cơ sở dữ liệu (Database Schema) đòi hỏi một mô hình tư duy thực thể khác biệt so với các ứng dụng truyền thống. Bài toán kỹ thuật cốt lõi đặt ra cho hệ quản trị cơ sở dữ liệu quan hệ MySQL ở đây là: Làm thế nào để lưu trữ đầy đủ thông tin định tuyến, quản lý phiên và lịch sử hội thoại nhưng vẫn đảm bảo tính chất *Zero-Knowledge* của máy chủ trung tâm.

Để giải quyết triệt để bài toán này, cấu trúc các bảng được phân rã thành hai nhóm dữ liệu có ranh giới bảo mật rõ rệt:

Dữ liệu định danh rõ ràng (Plaintext Metadata Fields): Bao gồm các trường khóa chính, khóa ngoại, UUID định danh thực thể, mốc thời gian ('timestamp') và trạng thái kết nối. Các dữ liệu này không thể mã hóa vì Core Server cần đọc trực tiếp dưới dạng tường minh để thực hiện các câu lệnh truy vấn liên kết ('JOIN'), lập chỉ mục ('Indexing') tăng tốc độ tìm kiếm và tối ưu hóa luồng định tuyến gói tin thời gian thực qua kết nối Socket.

Dữ liệu mật mã bất định (Cryptographic BLOB/Text Fields): Toàn bộ nội dung tin nhắn, thông tin khóa phiên trao đổi ('FileTransferKey', 'SessionKey') hay cấu trúc siêu dữ liệu của tệp tin tuyệt đối không được lưu trữ ở dạng văn bản rõ. Chúng được lưu dưới dạng chuỗi văn bản lớn ('TEXT') hoặc dữ liệu nhị phân ('BLOB') chứa bản mã (Ciphertext) đã qua bộ lọc mã hóa AES-256-GCM hoặc RSA-2048 từ phía Client, biểu diễn bằng cấu trúc mã hóa nhị phân sang văn bản Base64.

Đặc tả cấu trúc chi tiết các Bảng dữ liệu cốt lõi

Hệ thống thiết lập hai bảng dữ liệu trung tâm phục vụ trực tiếp cho mô hình truyền tin bảo mật và quản lý thực thể bao gồm: Bảng người dùng (users) và Bảng tin nhắn (messages).

Đặc tả cấu trúc Bảng người dùng (users): Bảng users chịu trách nhiệm lưu trữ thông tin tài khoản, chuỗi băm mật khẩu bảo mật cục bộ, trạng thái hiện diện thời gian thực và đặc biệt là khóa công khai RSA-2048 của từng cá nhân phục vụ cho chu trình phân phối khóa lai của hệ thống.

Bảng 4.1: Đặc tả các trường dữ liệu của bảng users

Tên trường	Kiểu dữ liệu	Ràng buộc	Ý nghĩa và Vai trò kỹ thuật
user_id	VARCHAR(36)	PRIMARY KEY	Mã định danh duy nhất của người dùng dưới định dạng UUIDv4, chống trùng lặp dữ liệu khi mở rộng.
username	VARCHAR(50)	UNIQUE, INDEX	Tên đăng nhập của người dùng, được lập chỉ mục để tối ưu tốc độ tìm kiếm danh bạ.
password_hash	VARCHAR(256)	NOT NULL	Chuỗi băm mật khẩu được xử lý bằng thuật toán PBKDF2 từ phía Client, Server dùng để đối chiếu xác thực.
rsa_public_key	TEXT	NOT NULL	Khóa công khai RSA-2048 dạng X.509 PEM Base64, dùng để các Client khác tải về mã hóa Session Key.
status	VARCHAR(15)	NOT NULL	Trạng thái hiện diện thời gian thực của tài khoản (ONLINE,

			OFFLINE, AWAY).
last_seen	TIMESTAMP	NOT NULL	Ghi nhận mốc thời gian tương tác cuối cùng của Socket Thread trước khi ngắt kết nối.
created_at	TIMESTAMP	DEFAULT NOW	Thời điểm khởi tạo tài khoản trên hệ thống.

Đặc tả cấu trúc Bảng tin nhắn (messages): Bảng messages đóng vai trò lưu trữ lịch sử hội thoại đã mã hóa đầu cuối và hàng đợi tin nhắn chờ (Pending Queue) cho các Client ngoại tuyến. Cấu trúc bảng phản ánh trực tiếp cấu hình gói tin E2EE đã thiết lập ở pha phân tích yêu cầu.

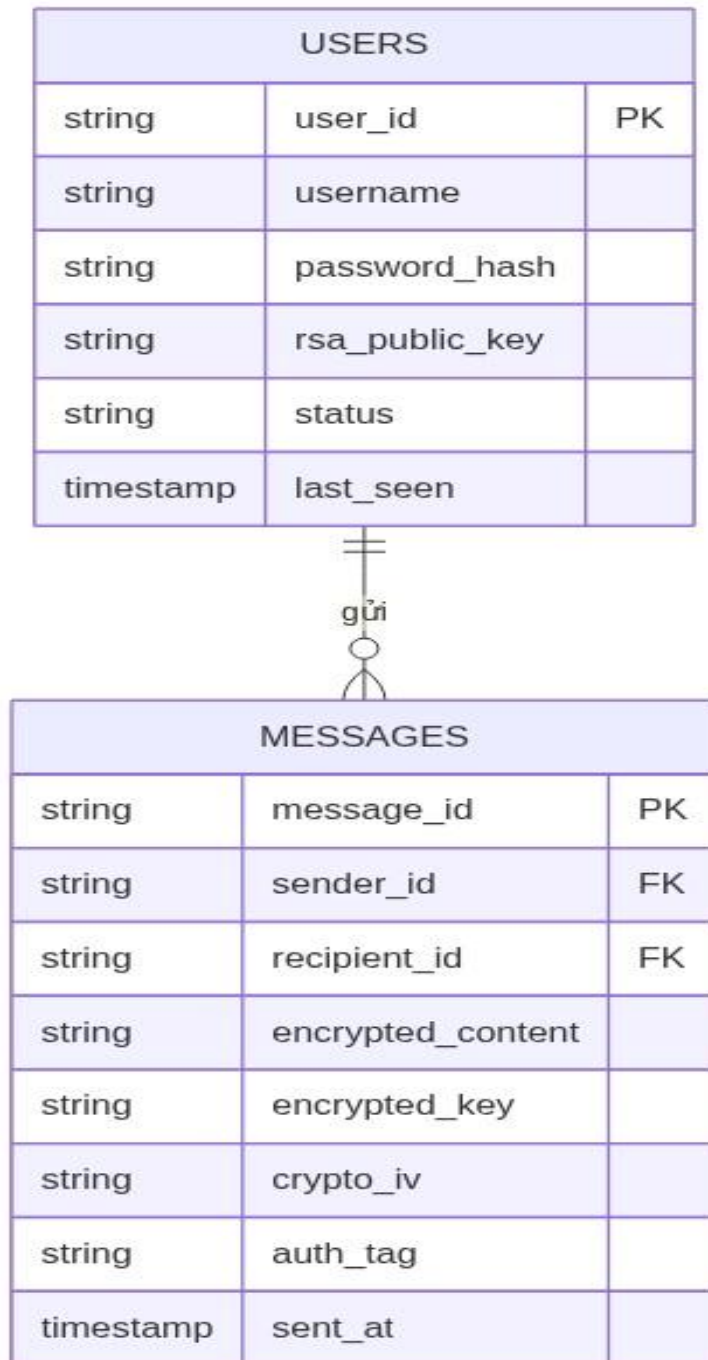
Bảng 4.2: Đặc tả các trường dữ liệu của bảng messages

Tên trường	Kiểu dữ liệu	Ràng buộc	Ý nghĩa và Vai trò kỹ thuật
message_id	VARCHAR(36)	PRIMARY KEY	Mã định danh duy nhất của gói tin nhắn (UUIDv4).
sender_id	VARCHAR(36)	FOREIGN KEY	Liên kết với user_id của bảng users, xác định chủ thể gửi bản tin (Lập chỉ mục B-Tree).
recipient_id	VARCHAR(36)	FOREIGN KEY	Liên kết với user_id của bảng users, phục vụ Core Server quét luồng định tuyến phân phối Socket.
encrypted_content	TEXT	NOT NULL	Bản mã nội dung tin nhắn (Ciphertext Base64) được mã hóa bằng thuật toán đối xứng AES-256 tại Client nguồn.

Tên trường	Kiểu dữ liệu	Ràng buộc	Ý nghĩa và Vai trò kỹ thuật
encrypted_key	TEXT	NOT NULL	Khóa phiên AES được mã hóa bằng RSA Public Key của người nhận (Ciphertext Base64).
crypto_iv	VARCHAR(44)	NOT NULL	Vector khởi tạo ngẫu nhiên (Initialization Vector) dùng cho hàm giải mã AES-GCM tại Client đích.
auth_tag	VARCHAR(44)	NOT NULL	Thẻ xác thực tính toàn vẹn (Authentication Tag) sinh ra từ chế độ mã hóa AEAD đối xứng.
delivery_status	VARCHAR(15)	NOT NULL	Trạng thái vòng đời gói tin (SENT, DELIVERED, READ).
sent_at	TIMESTAMP	NOT NULL	Mốc thời gian Client phát gói tin lên máy chủ trung tâm.

Mô hình Quan hệ và Liên kết Thực thể (Entity-Relationship Diagram)

Mô hình quan hệ tương tác logic, các ràng buộc toàn vẹn dữ liệu (ON DELETE RESTRICT /ON UPDATE CASCADE) và sơ đồ cấu trúc liên kết hình học giữa các bảng trong cơ sở dữ liệu được biểu diễn trực quan thông qua sơ đồ Entity-Relationship Diagram (ERD) dưới đây:



Hình 4.1: Sơ đồ Quan hệ Thực thể (Entity-Relationship Diagram) hệ thống Secure Chat

Để đảm bảo hiệu năng tối ưu khi số lượng bản ghi trong bảng messages tăng trưởng tuyến tính theo thời gian ($N \rightarrow \infty$), hệ thống thiết lập một Chỉ mục hỗn hợp (Composite Index) bao gồm cặp khóa (sender_id, recipient_id, sent_at). Chỉ mục này giúp rút ngắn độ phức tạp thuật toán tìm kiếm lịch sử hội thoại giữa hai cá nhân bất kỳ từ quét toàn bộ bảng ($O(N)$) xuống mức tiệm cận logarit ($O(\log N)$), đáp ứng điều kiện phản hồi dữ liệu tức thì lên giao diện bong bóng chat FlatLaf của Client.

3.6. Thiết kế Giao diện người dùng (UI/UX - Modern Aesthetic, Glassmorphism)

Thiết kế giao diện người dùng (UI) và trải nghiệm người dùng (UX) trong dự án Secure Chat đại diện cho một bước đột phá mặt kỹ thuật trên nền tảng Java Swing truyền thống. Bằng việc tích hợp thư viện lõi **FlatLaf (Flat Look and Feel)** kết hợp với các kỹ thuật tùy biến thành phần đồ họa cấp thấp (Low-level graphics rendering), hệ thống hiện thực hóa ngôn ngữ thiết kế thẩm mỹ hiện đại (Modern Aesthetic) và hiệu ứng kính mờ **Glassmorphism**. Thiết kế này không chỉ nâng cao giá trị thẩm mỹ trực quan mà còn tối ưu hóa luồng tương tác nhận thức (Cognitive workflow) của người dùng đối với các tính năng bảo mật, bảo vệ dữ liệu phức tạp.

1. Hiện thực hóa Hiệu ứng Kính mờ (Glassmorphism) trên Java Swing

Chứa dữ liệu đồ họa (JComponent):

Kỹ thuật Alpha Blending (Pha trộn kênh Alpha): Nhằm tạo độ trong suốt, hệ thống thực hiện ghi đè phương thức `paintComponent(Graphics g)` trong các lớp kế thừa từ `JPanel`. Bằng cách ép kiểu sang `Graphics2D`, thuộc tính `Composite` được thiết lập qua `AlphaComposite.getInstance(AlphaComposite.SRC_OVER, alpha)`. Chỉ số *alpha* được tinh chỉnh trong dải `[0.15, 0.30]` để phù hợp với từng phân tầng của giao diện.

Background Blur Filtering (Giải thuật làm mờ nền): Vì Java Swing không hỗ trợ sẵn bộ lọc Backdrop Blur, hệ thống triển khai giải thuật trích xuất vùng `BufferedImage` bên dưới panel kính mờ. Vùng ảnh này sau đó qua bộ lọc Gaussian Blur đa luồng (với bán kính nhiễu $r = 15\text{px}$) trước khi được vẽ ngược lại bề mặt đồ họa.

Specular Highlight Border (Đường viền phản xạ): Khung viền được vẽ thủ công bằng `BasicStroke(1.0f)` kết hợp mô hình `GradientPaint` chuyển tiếp từ trắng mờ sang trắng trong suốt. Kỹ thuật này giúp tái hiện chính xác hiện tượng khúc xạ ánh sáng tại các góc cạnh vật lý của tấm kính.

2. Kiến trúc Render Bong bóng Chat Gradient và Chế độ tối (Dark Mode)

Để tăng cường khả năng quét thông tin (scannable) và tính trực quan, phân hệ hiển thị hội thoại áp dụng các thuật toán hình học máy tính sau:

Bong bóng Chat Hình học mềm (Soft-geometry Bubbles): Thay vì các khối vuông thô cứng, các khung tin nhắn được kết xuất bằng `RoundRectangle2D.Float` với bán kính bo góc `16px`. Riêng các góc thắt ở đuôi hội thoại được xử lý đặc biệt bằng cách cộng gộp đa giác dựa trên cấu trúc `Path2D`.

2. Kiến trúc Render Bong bóng Chat Gradient và Chế độ tối (Dark Mode)

Phân hệ hiển thị hội thoại thời gian thực áp dụng các giải thuật hình học máy tính để tối ưu tính scannable (dễ quét thông tin) cho luồng tương tác trực quan:

Bong bóng Chat Hình học mềm (Soft-geometry Chat Bubbles): Các khung chứa tin nhắn không sử dụng các khối chữ nhật thô cứng mà được vẽ thông qua cấu trúc lớp hình học `RoundRectangle2D.Float` với các góc bo tròn mềm mại (`$R_{corner} = 16 \text{ extpx}`). Riêng góc bo tại vị trí nút thắt của tin nhắn (đuôi hội thoại gửi/nhận) được xử lý loại trừ bằng cách

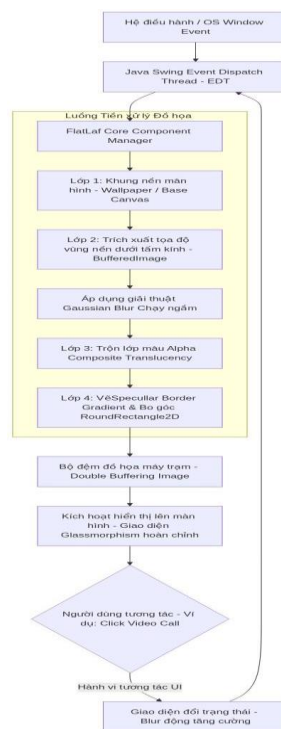
cộng gộp đa giác dựa trên cấu trúc `Path2D`.

Tô màu Gradient động tuyến tính (Linear Gradient Engine): Đối với tin nhắn của chính người gửi (Sender), nền bong bóng chat được áp dụng giải thuật `LinearGradientPaint` dịch chuyển màu sắc theo hướng vector góc 45° từ mã màu chủ đạo `#3a7bd5` đến `#3a6073`. Hệ thống tự động tính toán lại ma trận màu sắc dựa trên chiều rộng co giãn thực tế của bong bóng chat nhằm ngăn chặn lỗi vỡ dải màu (Color banding).

Dark Mode và Đồng bộ hóa Phần cứng (FlatLaf Architecture): Dự án tích hợp thư viện FlatLaf phiên bản cao cấp để truy cập vào các API hệ điều hành nội bộ. Hệ thống lắng nghe sự thay đổi giao diện của OS (Windows/macOS Dark Theme Registry) thông qua thư viện `FlatPreferences`, tự động chuyển đổi cấu hình bảng mã màu (Color Palette) từ `FlatDarkLaf` sang `FlatLightLaf` mà không yêu cầu người dùng khởi động lại ứng dụng.

3. Sơ đồ Kiến trúc phân tầng đồ họa và Luồng xử lý sự kiện UI

Để render một bảng điều khiển ứng dụng Secure Chat phức tạp chứa hiệu ứng Glassmorphism mà không gây suy giảm tốc độ xử lý khung hình (Frame rate Drop), hệ thống thực thi theo kiến trúc kết xuất phân lớp và kỹ thuật đệm đôi (Double Buffering) được mô tả qua sơ đồ dưới đây:



Hình 25: Luồng xử lý kỹ thuật phân tầng đồ họa khi kết xuất hiệu ứng Glassmorphism trên Client.

4. Các thông số Kỹ thuật và Ma trận chỉ số Thiết kế UI/UX

Các thông số thiết kế hình học, tỷ lệ khoảng cách, sắc độ màu sắc và phân tầng giao diện hiển thị được quy định chi tiết và nghiêm ngặt trong Bảng 1 nhằm đảm bảo tính đồng nhất hệ thống.

Thông số cấu hình kỹ thuật và Định chuẩn hình học lớp UI/UX

Thành phần / Yếu tố	Thông số kỹ thuật	Mục đích / Ý nghĩa
Màu sắc giao diện (Color Scheme)	Surface: #1e1e1e Primary: #0b84ff	Tuân thủ quy chuẩn Material Design, giảm thiểu cường độ bức xạ ánh sáng xanh, chống mỏi mắt cho người dùng khi giao tiếp kéo dài.
Kênh Alpha kính mờ (Glass Translucency)	0.18f (18% opacity)	Đảm bảo nhìn xuyên thấu được cấu trúc ảnh nền hoặc các lớp màu đa tầng phía sau mà không làm lu mờ nội dung văn bản text phía trước.
Bán kính nhiễu Gaussian (Gaussian Blur)	15px sigma filter	Triệt tiêu hoàn toàn các chi tiết sắc nhọn của hình ảnh nền bên dưới tấm kính, cô lập và tăng cường độ tập trung nhận thức thị giác vào các nút chức năng.
Kiểu chữ (Typography System)	Font: Inter / Segoe UI Size: 13pt (Body), 16pt (H2)	Font chữ không chân (Sans-serif) hình học hiện đại, có cự ly pixel và khoảng cách dòng (Line-height = 1.4) tối ưu cho việc đọc tin nhắn.

Tốc độ phản hồi hiệu ứng (UI Transition Animation)	250ms (Linear Interpolation)	Áp dụng bộ hàm biến đổi toán học tuyến tính cho các chuyển động trượt mượt mà của thanh Sidepanel hoặc bong bóng chat AI mà không tạo độ trễ cảm giác.
---	---------------------------------	--

5. Thiết kế luồng UX đặc thù cho các tính năng Bảo mật và Trợ lý AI

Trải nghiệm người dùng đối với các tính năng kỹ thuật cao được tối ưu hóa thông qua các kịch bản tương tác ẩn (Micro-interactions):

Luồng UX chỉ thị trạng thái Mã hóa đầu cuối (E2EE Indicator): Tại thanh tiêu đề (Header) của mỗi cuộc hội thoại, hệ thống thiết lập một biểu tượng "Khóa bảo mật" sử dụng màu xanh ngọc ('#34c759'). Khi nhấn vào biểu tượng này, một panel kính mờ sẽ mở ra, hiển thị mã băm công khai (Public Key Fingerprint) của hai Client dưới dạng chuỗi hex phân đoạn 4 ký tự. Cơ chế này giúp người dùng kiểm thử và xác minh tính toàn vẹn của kênh truyền mà không cần có kiến thức chuyên sâu về mã hóa RSA-2048.

Luồng UX hiển thị nội dung sinh văn bản của AI (Ollama Token Flux): Để tối ưu hóa trải nghiệm tương tác với Local AI, bong bóng chat dành riêng cho trợ lý ảo Ollama áp dụng hiệu ứng mờ viền chuyển tiếp (Fading Gradient Edge). Khi các token được sinh ra và đẩy về dạng dòng từ API, bong bóng chat tự động cuộn (Auto-scroll) theo thuật toán giảm chấn (Damping Algorithm). Thuật toán này tính toán khoảng cách cuộn trượt giảm dần theo thời gian, giúp chuyển động cuộn của khung chat mịn màng, không bị giật lag đột ngột khi tốc độ sinh từ của mô hình tăng cao.

CHƯƠNG 4: THIẾT KẾ GIAO DIỆN NGƯỜI DÙNG (UI/UX)

4.1. Nguyên tắc thiết kế (Design Principles)

Hệ thống Secure Chat được thiết kế tuân thủ nghiêm ngặt các nguyên tắc của Material Design và triết lý UI/UX hiện đại. Bố cục ứng dụng tập trung vào sự tối giản (Minimalism), sử dụng không gian trắng (Whitespace) hợp lý và phân cấp thông tin rõ ràng. Các nguyên tắc cốt lõi bao gồm:

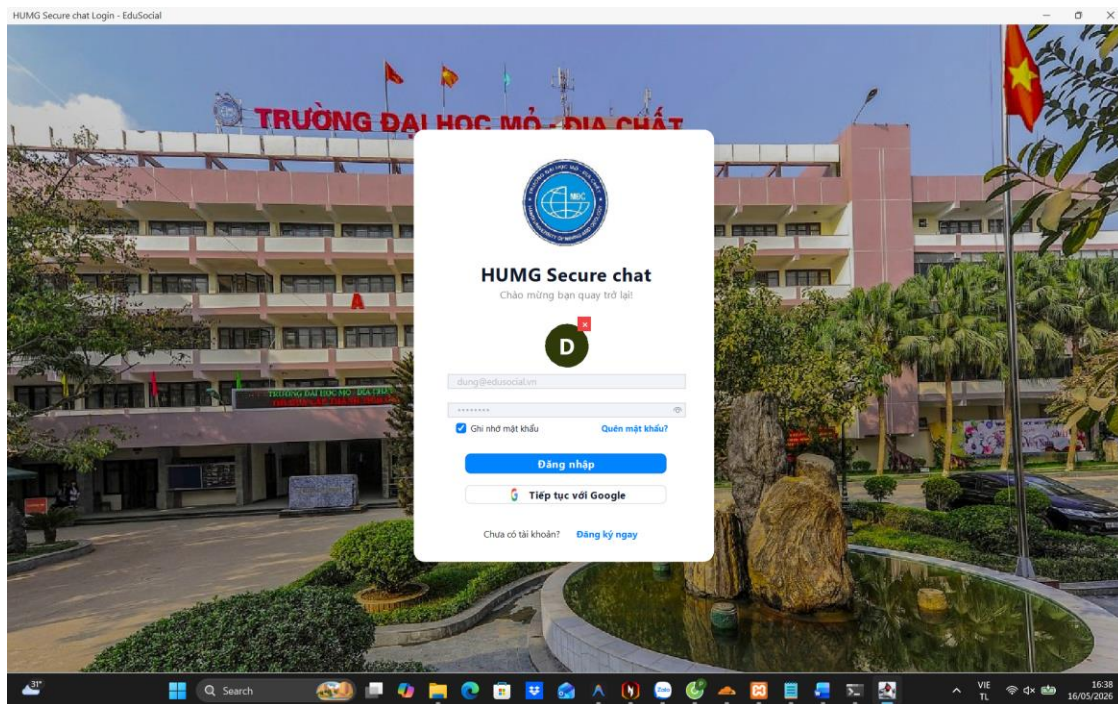
Security-first (Ưu tiên bảo mật): Các tính năng E2EE, OTP, 2FA được thiết kế nổi bật để xây dựng lòng tin.

Consistency (Tính nhất quán): Màu sắc, typography và các component (Nút bấm, Modal, Form) được đồng bộ trên toàn hệ thống.

Feedback (Phản hồi hệ thống): Cung cấp các thông báo trạng thái rõ ràng (Loading, Success, Error, Empty State) cho mọi thao tác của người dùng.

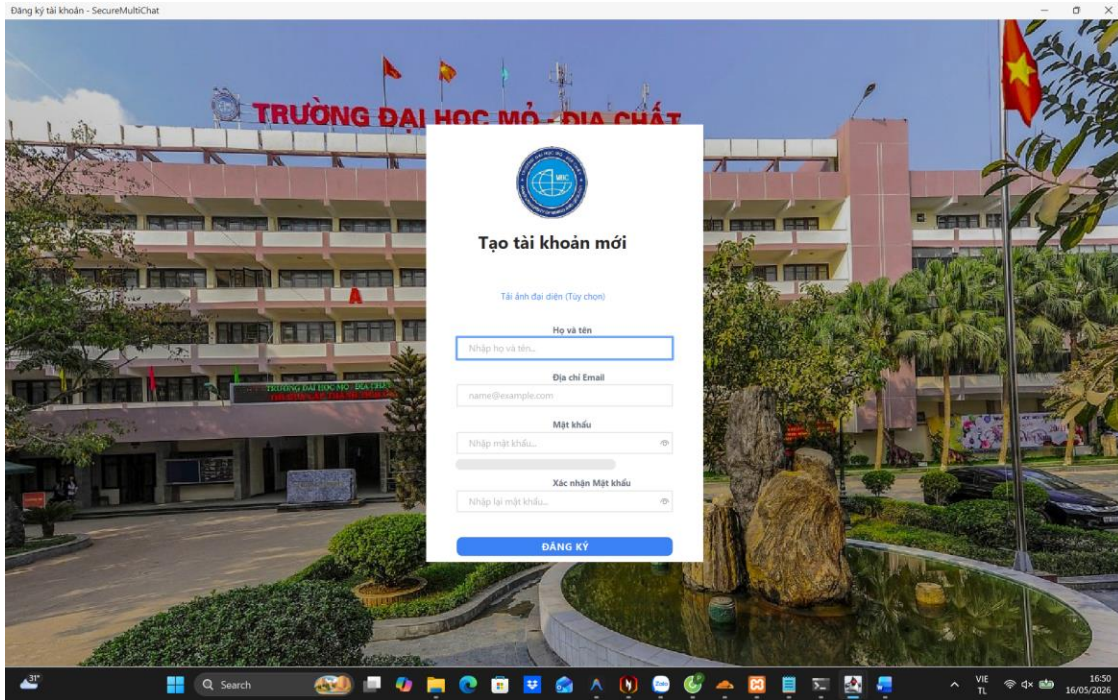
Dưới đây là phân tích chi tiết thiết kế giao diện của các phân hệ chức năng cốt lõi trong hệ thống.

4.2. Phân hệ Xác thực và Quản lý Tài khoản (Auth & Profiles)



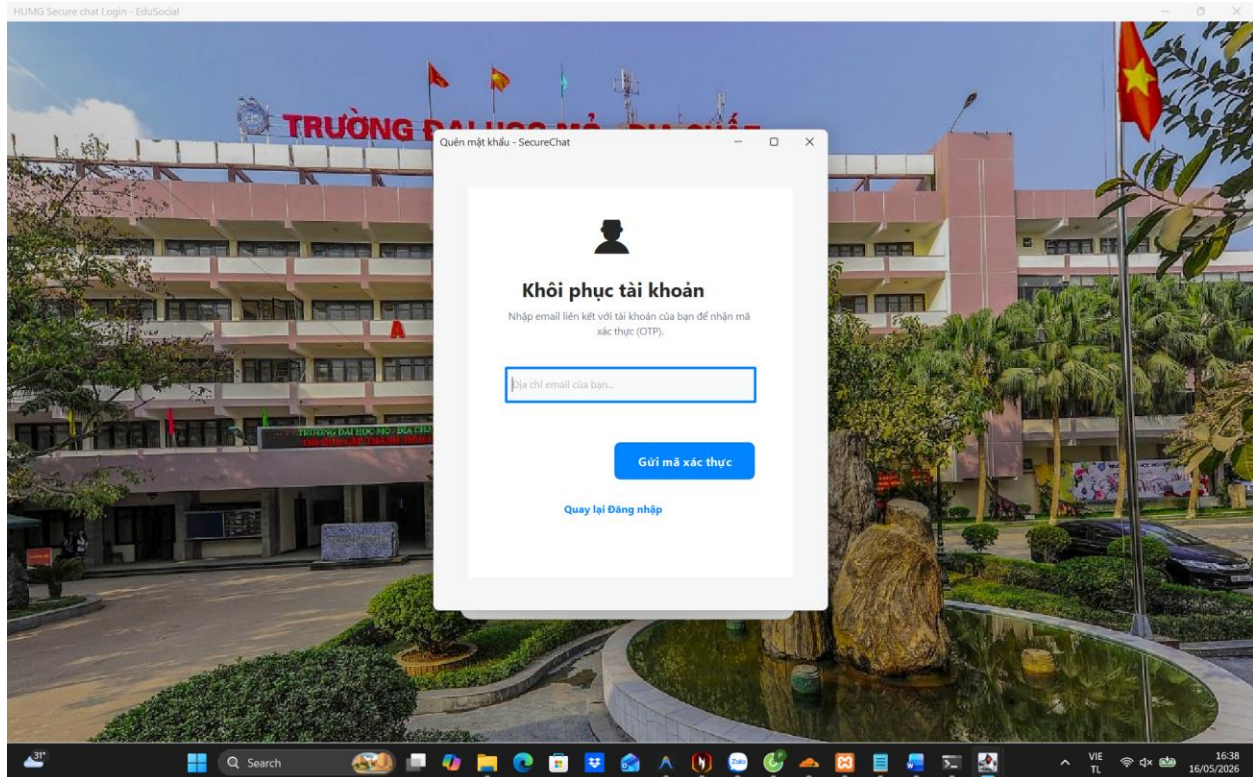
Hình 4.1: Giao diện Đăng nhập hệ thống (Login)

Mô tả UI: Bố cục chia đôi màn hình (Split-screen layout). Một bên là hình ảnh minh họa tạo ấn tượng thị giác, bên còn lại là Form đăng nhập tối giản. Form bao gồm các trường Email và Mật khẩu (có chức năng ẩn/hiện), nút CTA “Đăng nhập” nổi bật. Tích hợp liên kết “Quên mật khẩu?” và điều hướng sang trang Đăng ký.



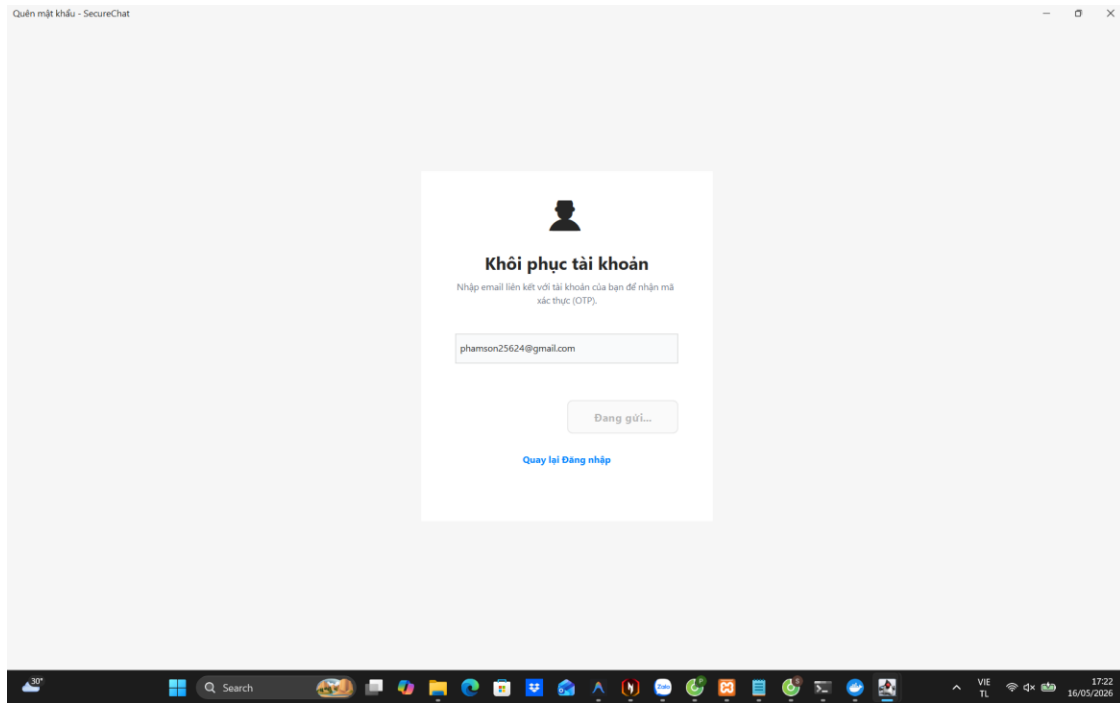
Hình 4.2: Giao diện Đăng ký tài khoản (Register)

Mô tả UI: Form đăng ký yêu cầu nhập thông tin cơ bản. Hệ thống tích hợp Real-time Validation (kiểm tra tính hợp lệ ngay khi gõ) để đảm bảo mật khẩu đạt chuẩn bảo mật (độ dài, ký tự đặc biệt) trước khi gửi request, tối ưu hóa trải nghiệm.



Hình 4.3: Giao diện Xác thực Email (OTP Verification)

Mô tả UI: Thiết kế tập trung (Focused UI) với ô nhập mã OTP lớn. Có tính năng đếm ngược thời gian (Countdown timer) để yêu cầu gửi lại mã, phòng chống spam API từ phía client.



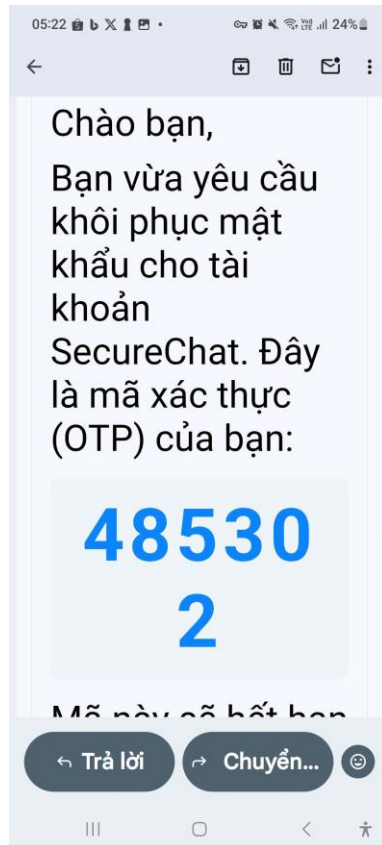
Hình 4.4: Giao diện Yêu cầu khôi phục tài khoản

Mô tả UI: Giao diện thẻ (Card UI) tối giản, yêu cầu nhập Email liên kết. Trạng thái nút bấm hiển thị “Đang gửi...” cho thấy hệ thống đang xử lý request gọi API ngầm.



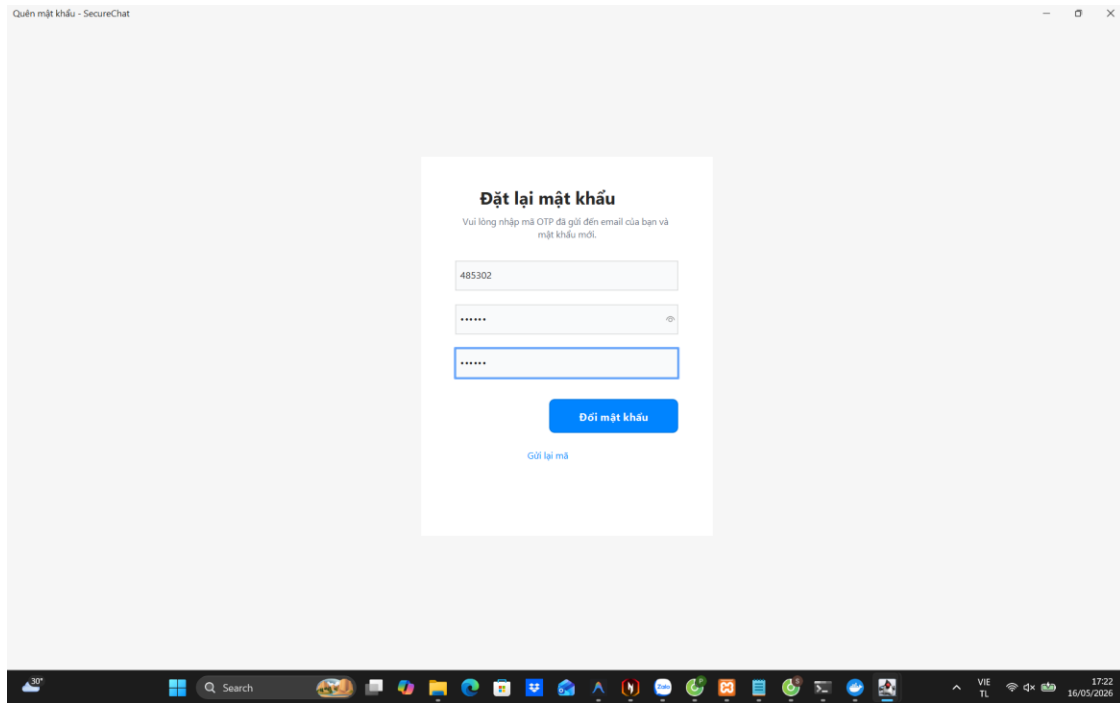
Hình 4.5: Email thông báo yêu cầu khôi phục mật khẩu

Mô tả UI: Tiêu đề email rõ ràng “Khôi phục mật khẩu tài khoản SecureChat” giúp định danh hệ thống, tạo sự tin tưởng.



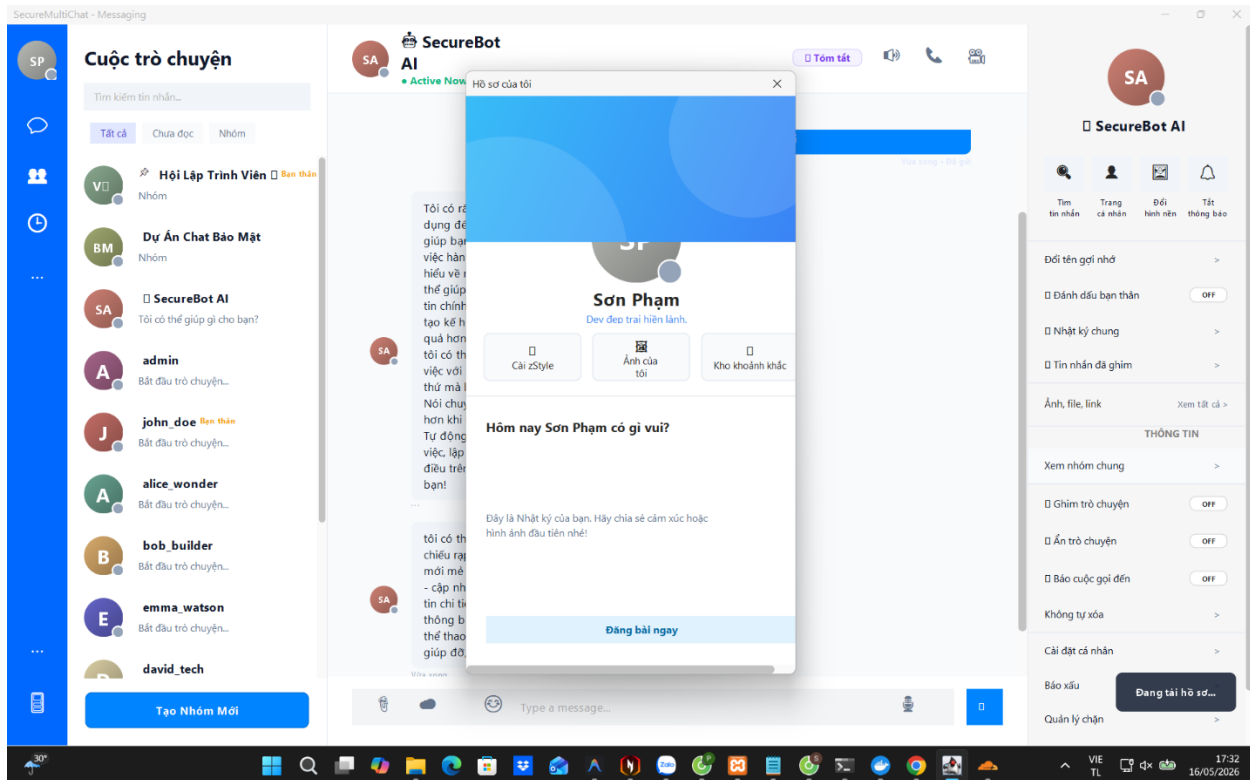
Hình 4.6: Email cung cấp mã xác thực OTP

Mô tả UI: Cấu trúc email bảo mật gửi kèm mã OTP 6 chữ số được in đậm, kích thước lớn giúp người dùng dễ sao chép. Có kèm cảnh báo thời gian hết hạn.



Hình 4.7: Giao diện Đặt lại mật khẩu mới

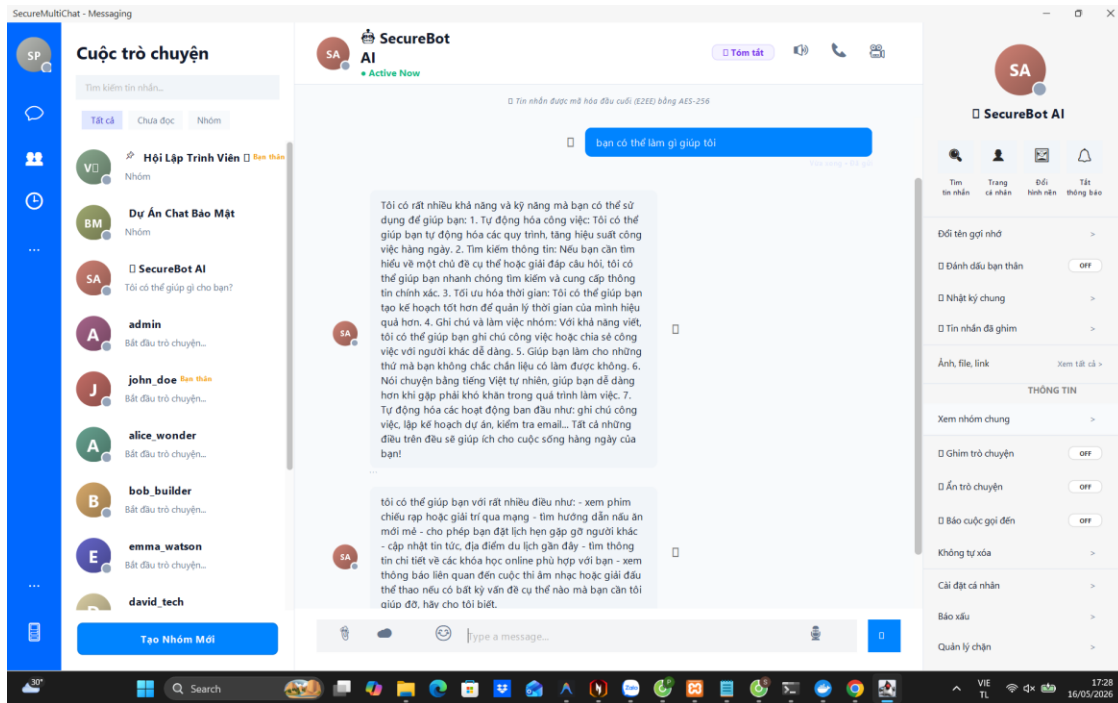
Mô tả UI: Form nhập mã OTP kèm 2 trường: Mật khẩu mới và Xác nhận lại. Có icon “mắt” để xem/ẩn mật khẩu. Nút CTA nổi bật màu xanh dương.



Hình 4.8: Giao diện Cài đặt Hồ sơ Cá nhân (User Profile)

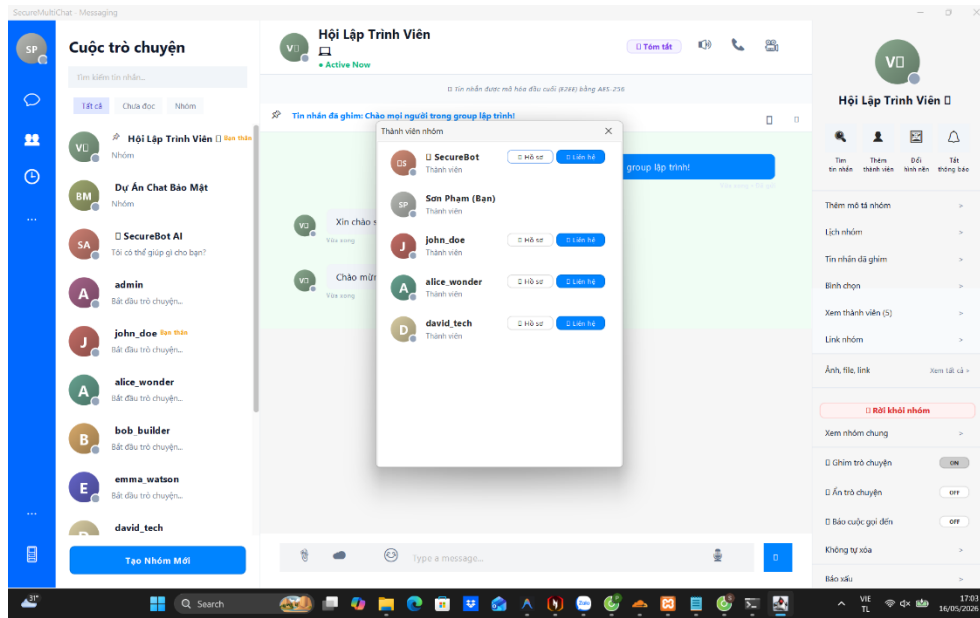
Mô tả UI: Màn hình quản lý Avatar, Họ tên và Tiểu sử. Sử dụng công tắc (Toggle Switch) cho các thiết lập: Thông báo, Giao diện Sáng/Tối.

4.3. Phân hệ Nhắn tin và Tương tác cốt lõi (Core Messaging & AI)



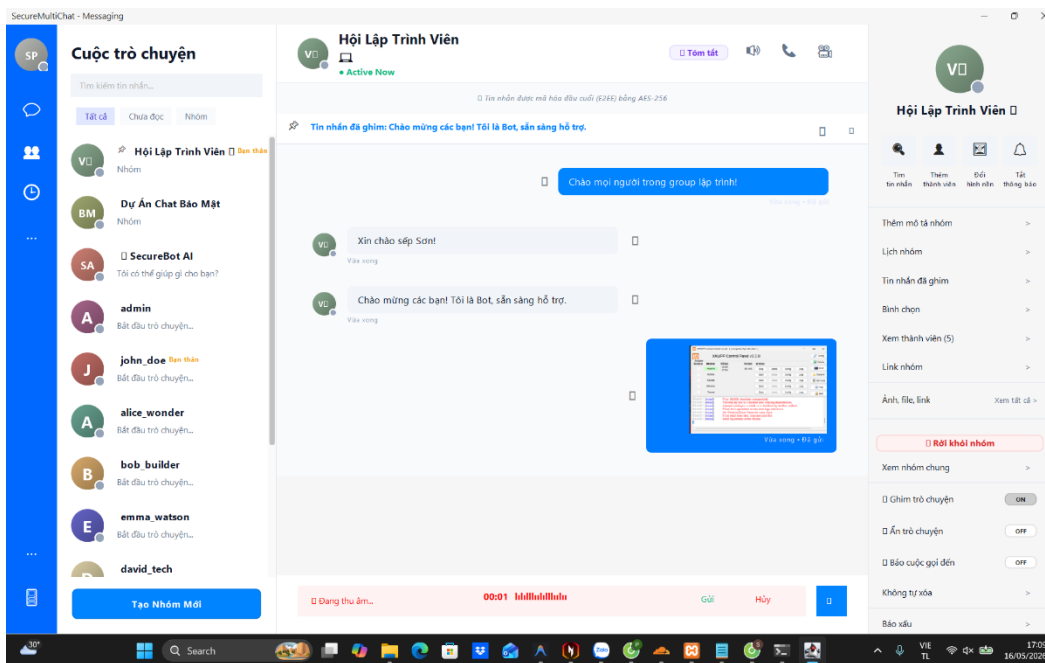
Hình 4.9: Giao diện Nhắn tin chính và Tương tác AI Bot

Mô tả UI: Bố cục 3 cột (3-column layout). Cột trái: Điều hướng và danh sách chat. Cột giữa: Nội dung chat (có thông báo E2EE bảo mật). Cột phải: Thông tin chi tiết và cài đặt đoạn chat. Hỗ trợ thanh nhập liệu đa năng.



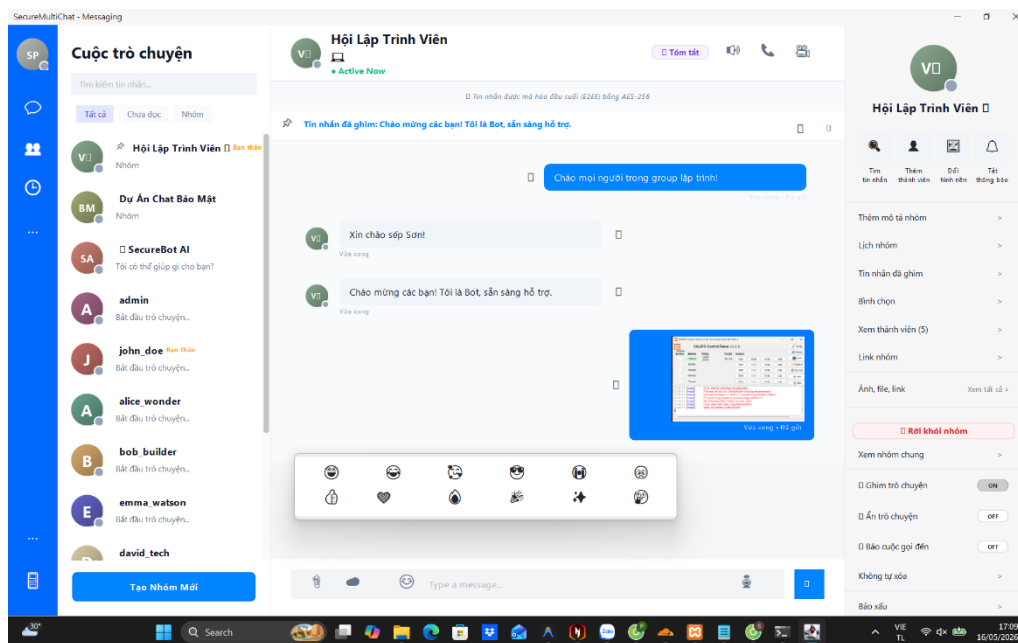
Hình 4.10: Giao diện Nhắn tin Nhóm (Group Chat)

Mô tả UI: Header hiển thị số lượng thành viên. Cột phải liệt kê danh sách thành viên, cung cấp quyền quản trị cho Group Admin (Xóa, Giải tán). Nút Gọi Video tích hợp sẵn.



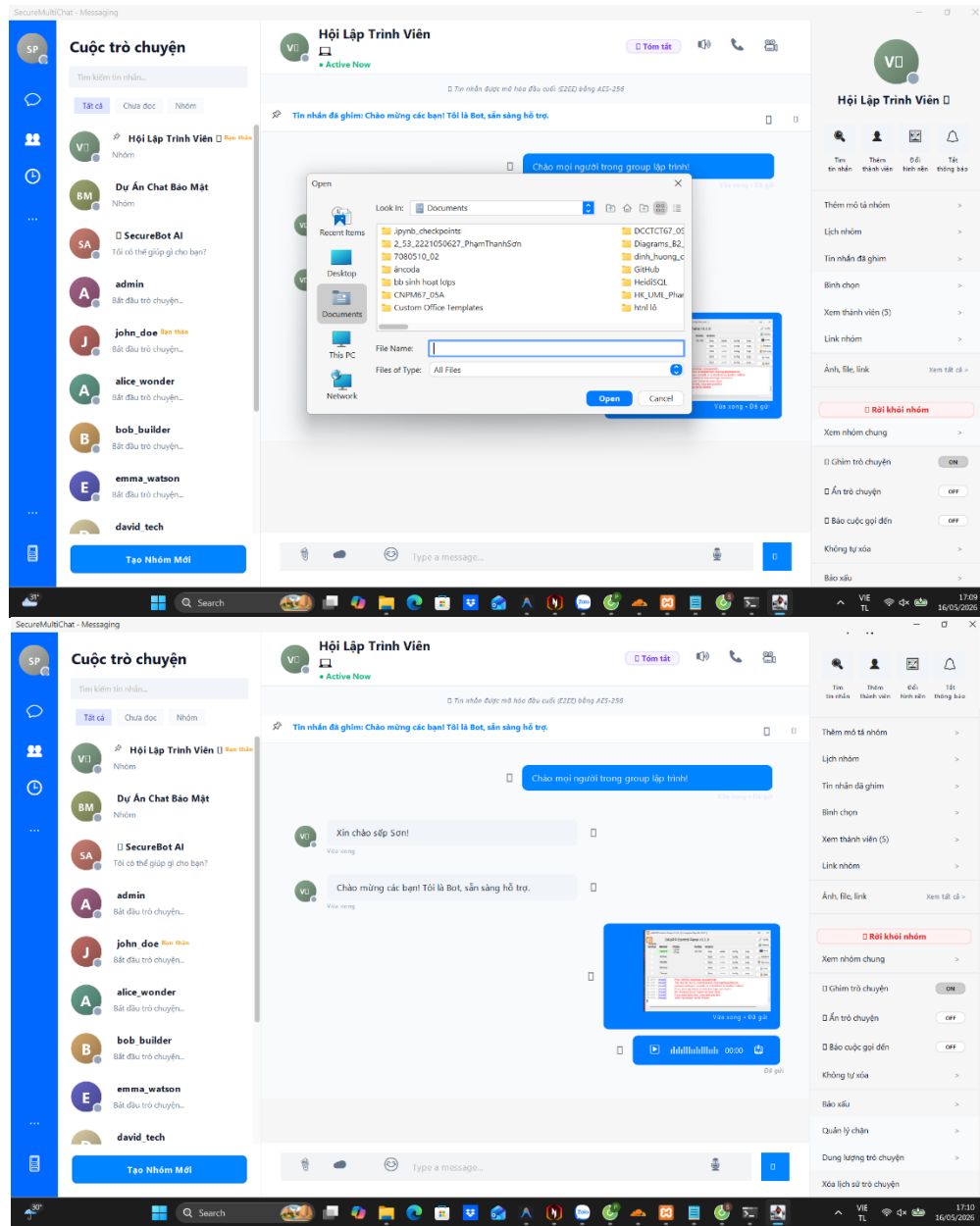
Hình 4.11: Giao diện Tin nhắn Ghi âm (Voice Note)

Mô tả UI: Bong bóng chat chứa biểu đồ sóng âm (Audio Waveform) phản ánh cường độ âm thanh, kèm nút Play/Pause và bộ đếm thời gian.



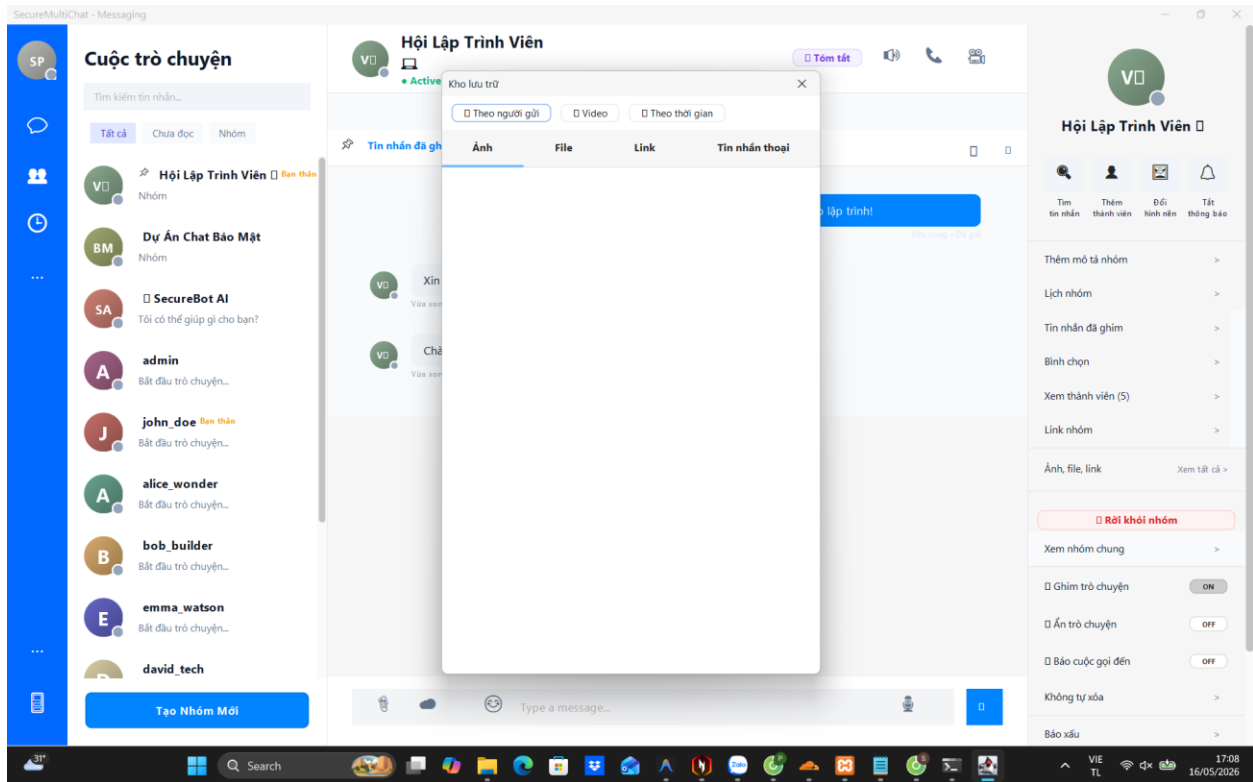
Hình 4.12: Giao diện Tương tác Biểu tượng cảm xúc (Reactions)

Mô tả UI: Thanh công cụ nổi (Floating Bar) xuất hiện khi nhấn giữ tin nhắn. Emoji được gắn nhỏ ở góc dưới bong bóng chat kèm bộ đếm số lượng.



Hình 4.13: Giao diện Truyền tải File đính kèm mã hóa

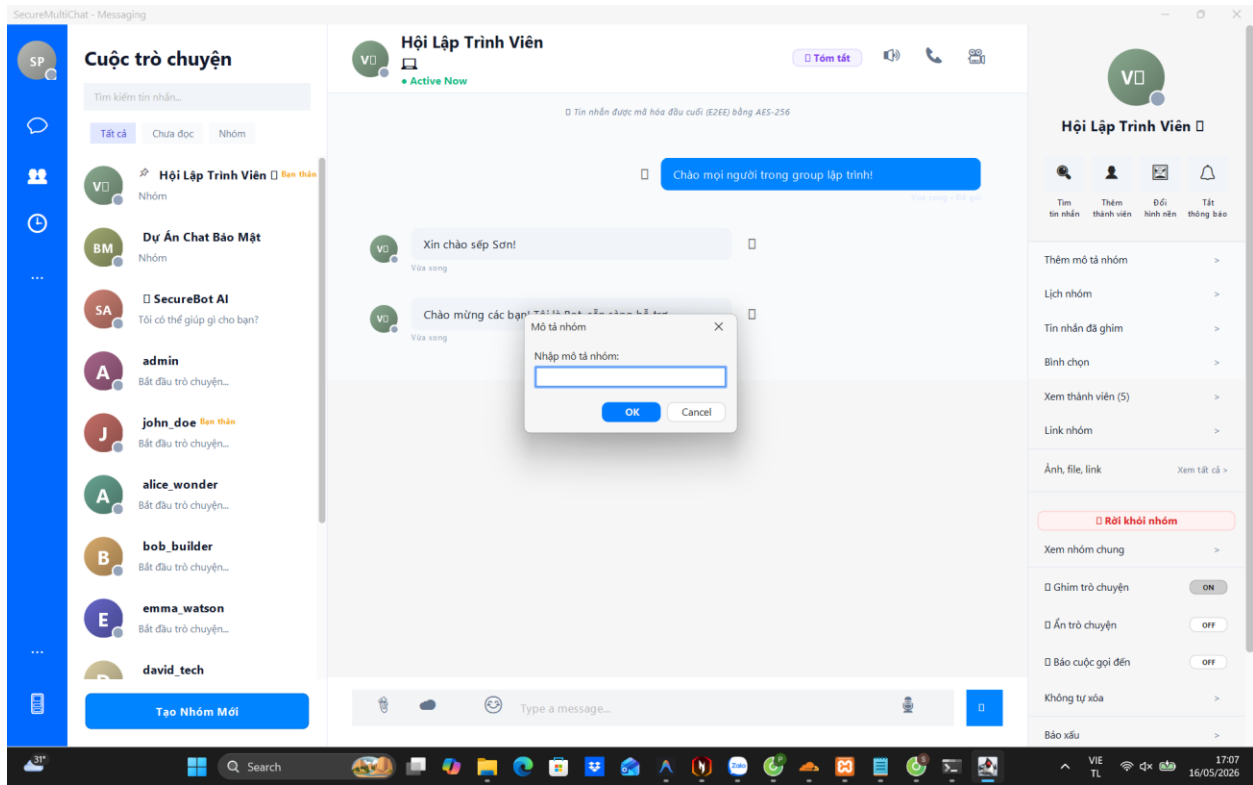
Mô tả UI: File (PDF, Word, Zip) hiển thị dạng thẻ (File Card) kèm Icon, Tên, Dung lượng. Có dòng chữ “Mã hóa E2EE” khẳng định dữ liệu được bảo vệ.



Hình 4.14: Cửa sổ xem trước Hình ảnh (Lightbox)

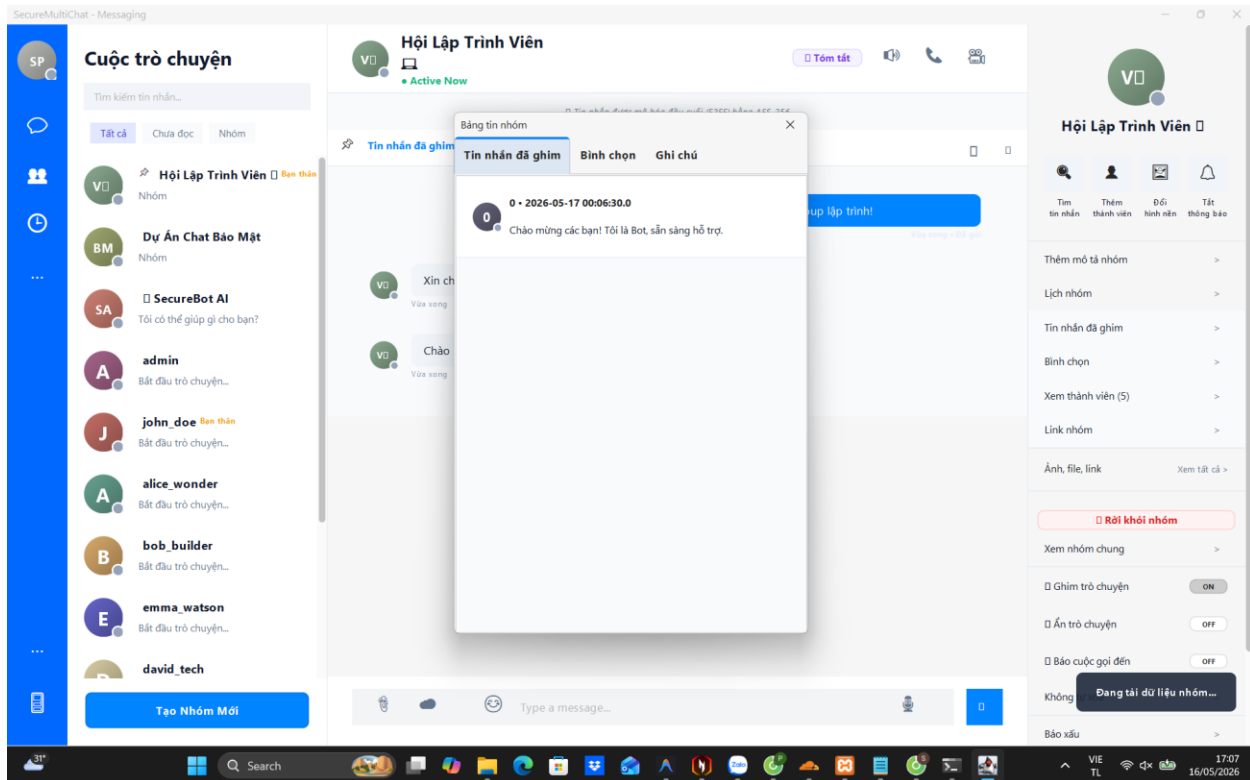
Mô tả UI: Chế độ Lightbox làm tối nền xung quanh (Dark Backdrop), tập trung 100% thị giác vào bức ảnh đang xem.

4.4. Phân hệ Tiện ích Chat và Quản lý Cuộc trò chuyện



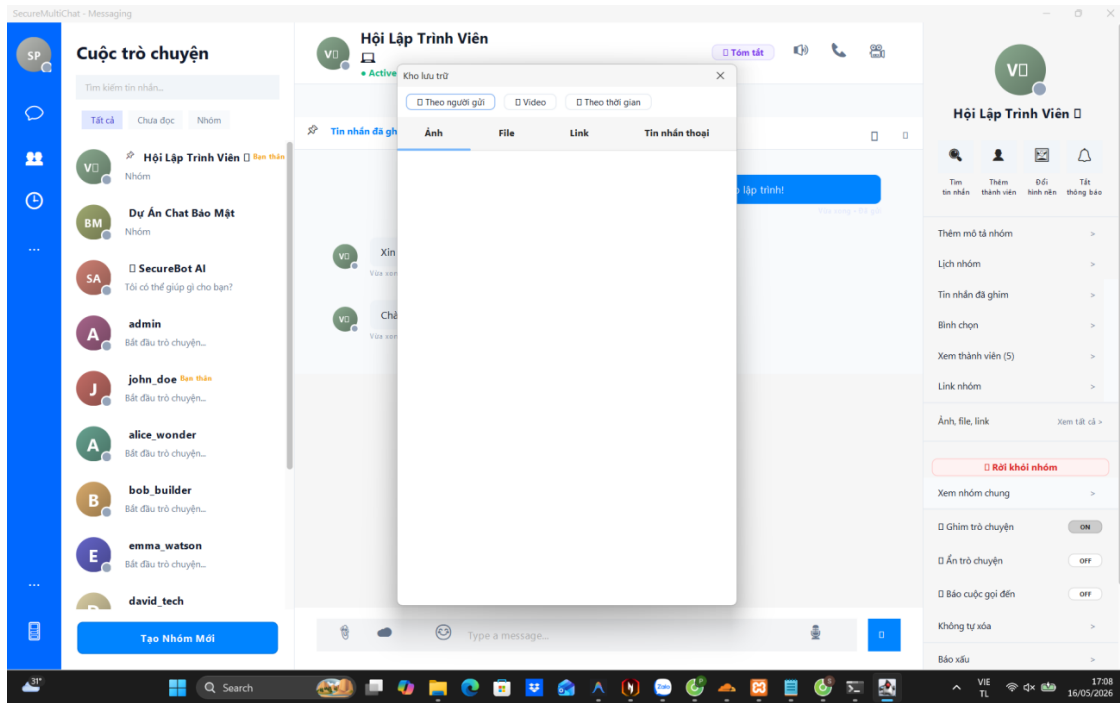
Hình 4.15: Khung Quản lý Thông tin Nhóm

Mô tả UI: Right Sidebar dùng cấu trúc Accordion (danh sách thả xuống) để nhóm chức năng, tiết kiệm không gian.



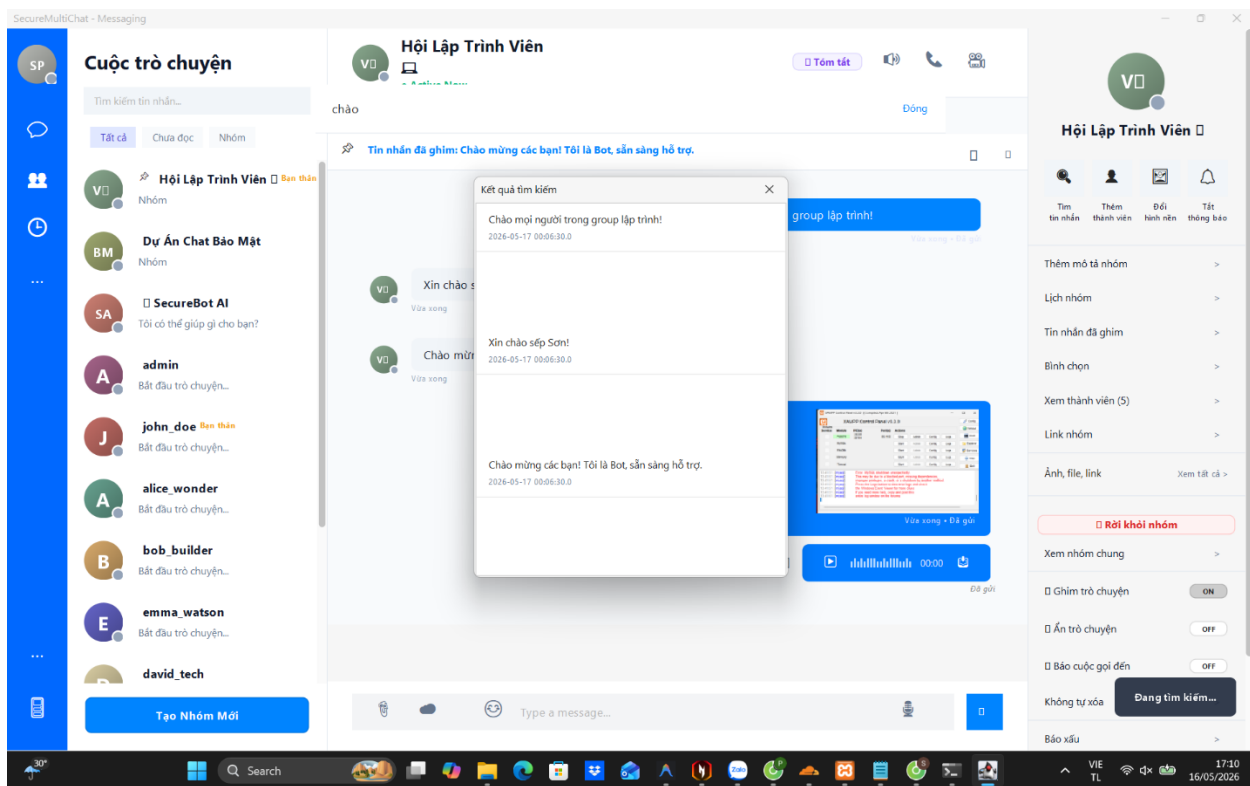
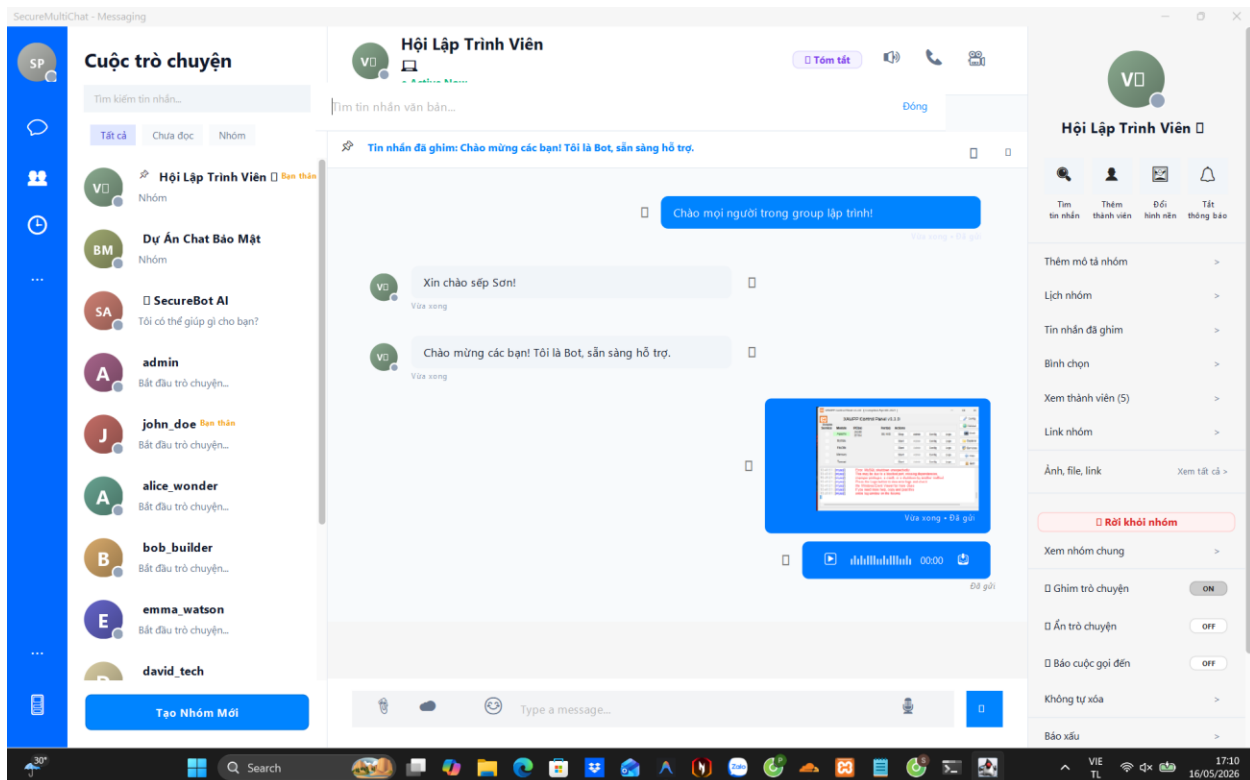
Hình 4.16: Quản lý Tin nhắn đã ghim (Pinned Messages)

Mô tả UI: Click vào tin nhắn ghim sẽ tự động cuộn (scroll) màn hình về vị trí tin nhắn gốc, tối ưu tra cứu.



Hình 4.17: Thư viện Đa phương tiện (Media Gallery)

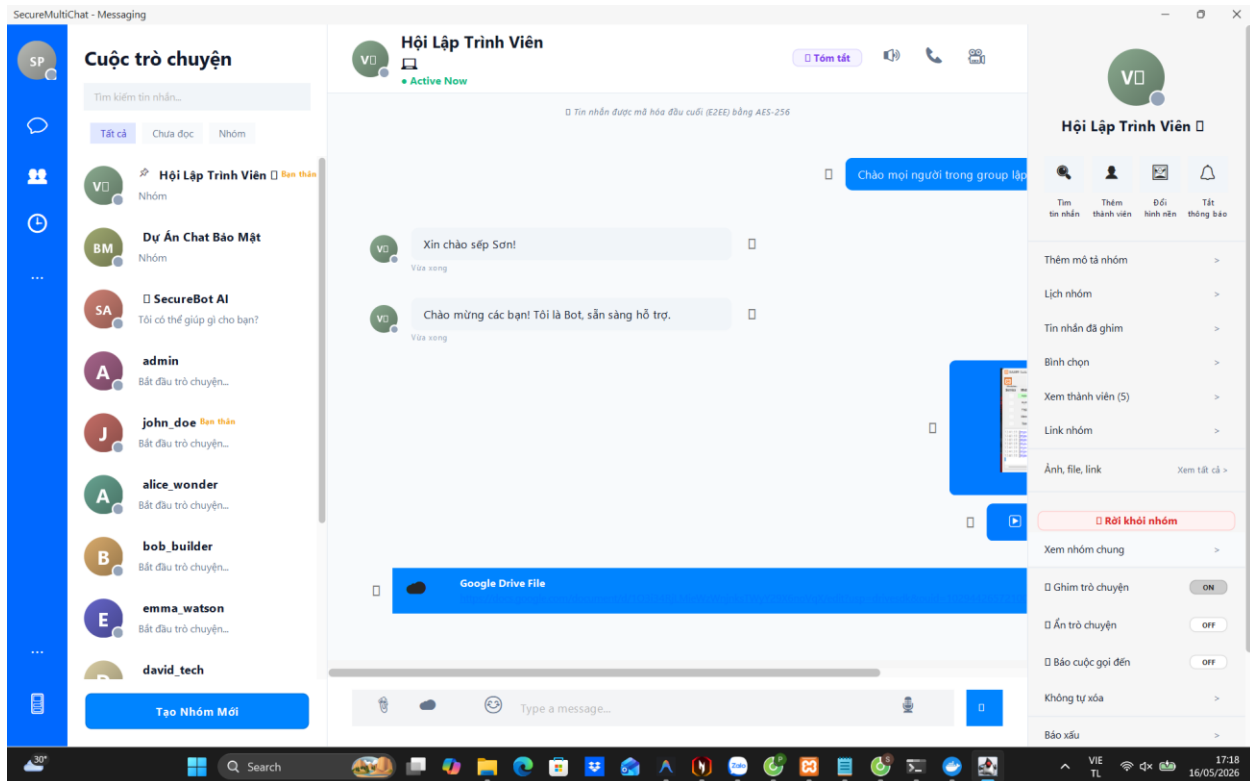
Mô tả UI: Bố cục dạng lưới (Grid Layout) hiển thị Thumbnail ảnh/video, tối đa hóa số lượng trên một diện tích.



Hình 4.18: Tính năng Tìm kiếm trong Chat (In-chat Search)

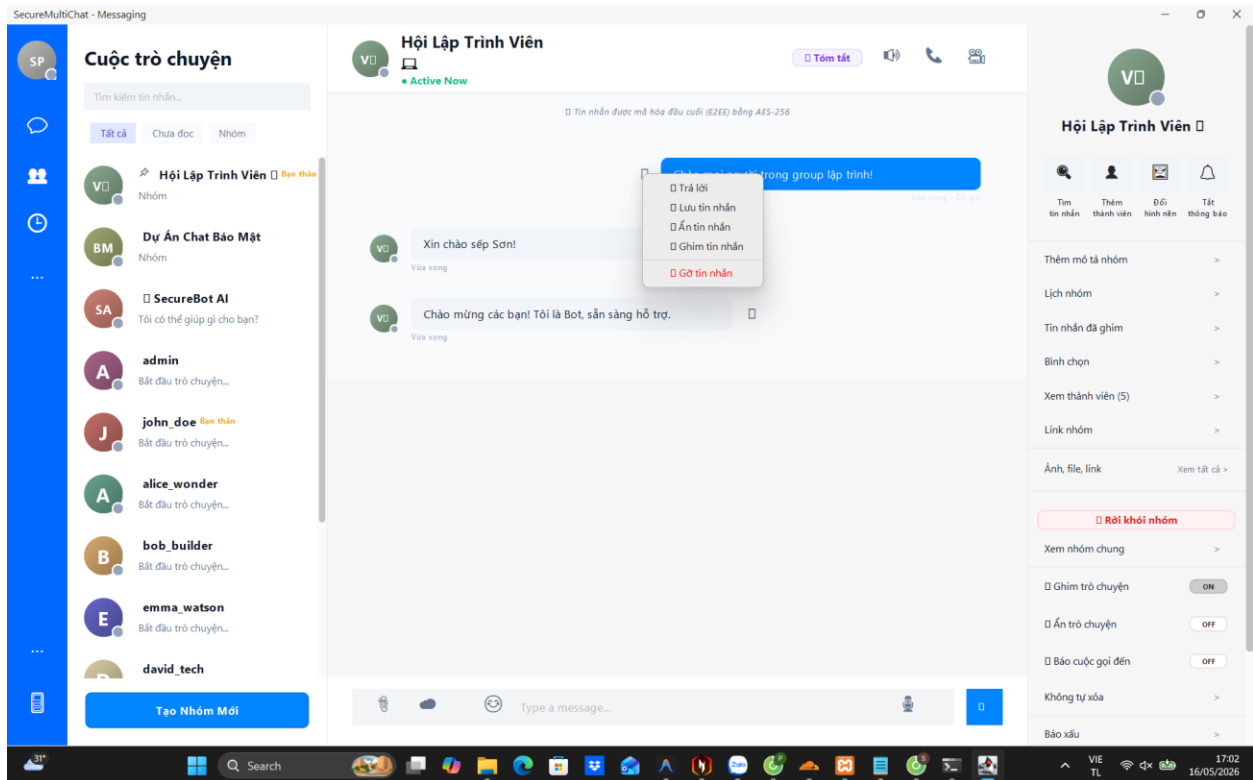
Mô tả UI: Kết quả trả về dạng danh sách. Áp dụng kỹ thuật Highlight Keyword (tô sáng từ khóa) giúp mắt quét nhanh ngữ cảnh.

Hình 4.24: Quản lý Liên kết đã chia sẻ (Shared Links)



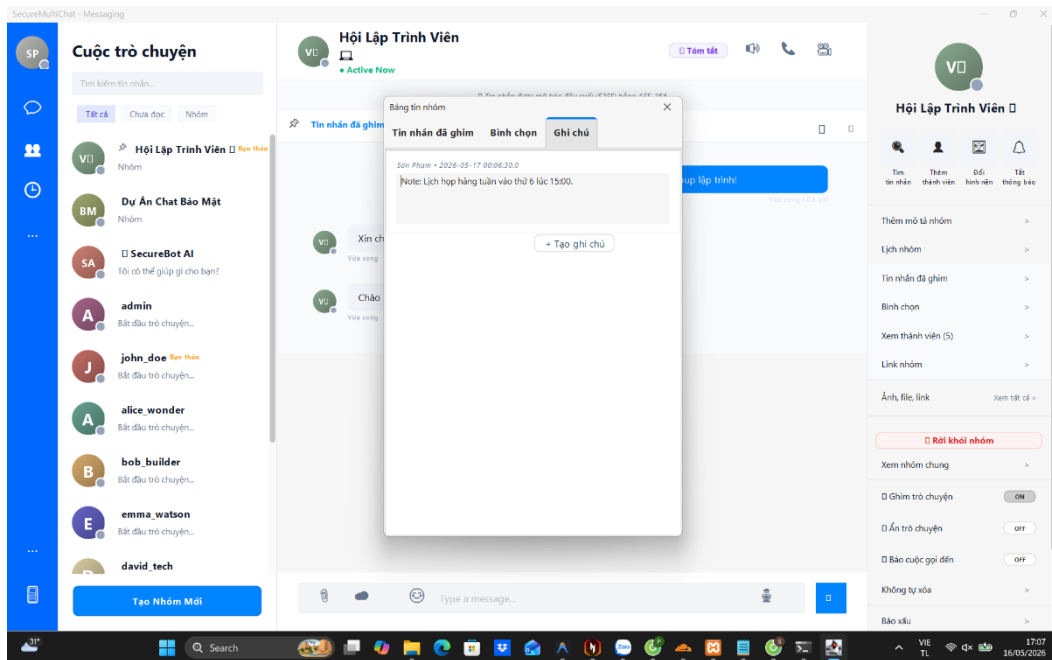
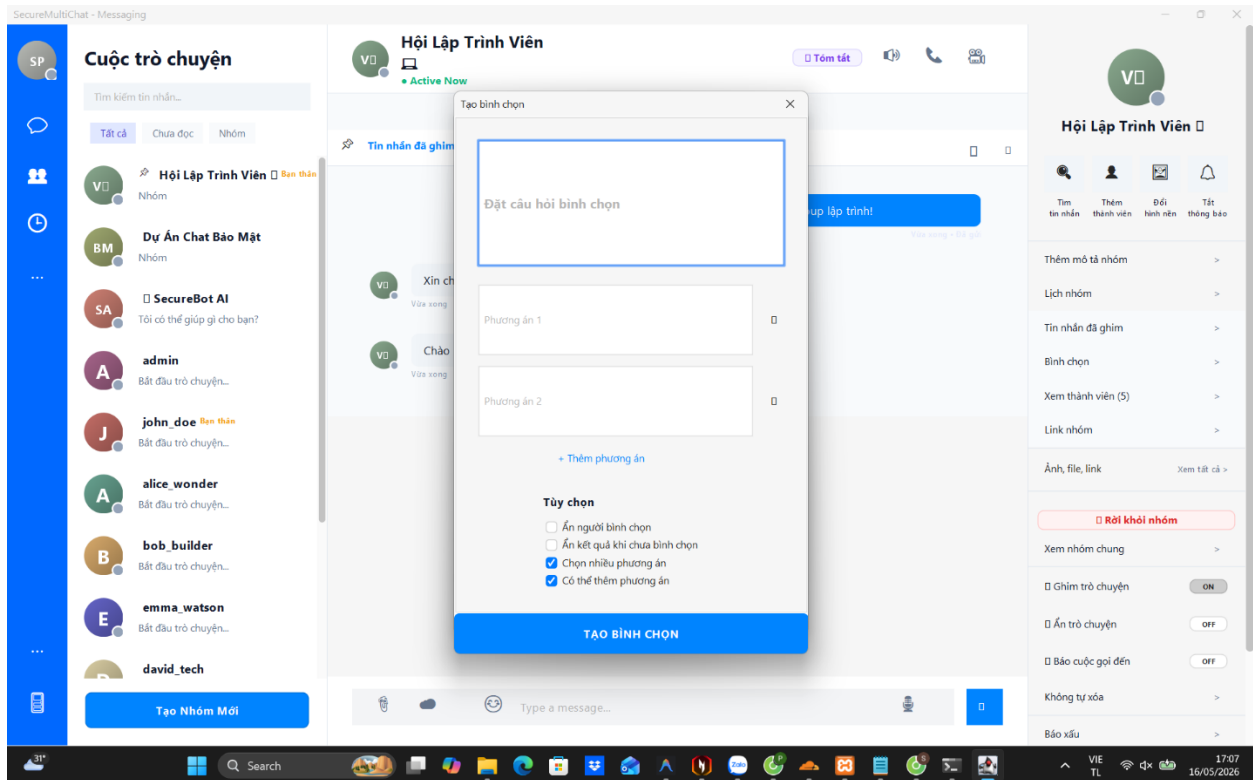
Hình 4.19: Quản lý Liên kết đã chia sẻ (Shared Links)

Mô tả UI: Cắt ngắn URL quá dài (Truncate) để đảm bảo layout gọn gàng, hiển thị kèm favicon trang web.



Hình 4.20: Giao diện Chỉnh sửa tin nhắn (Edit Message)

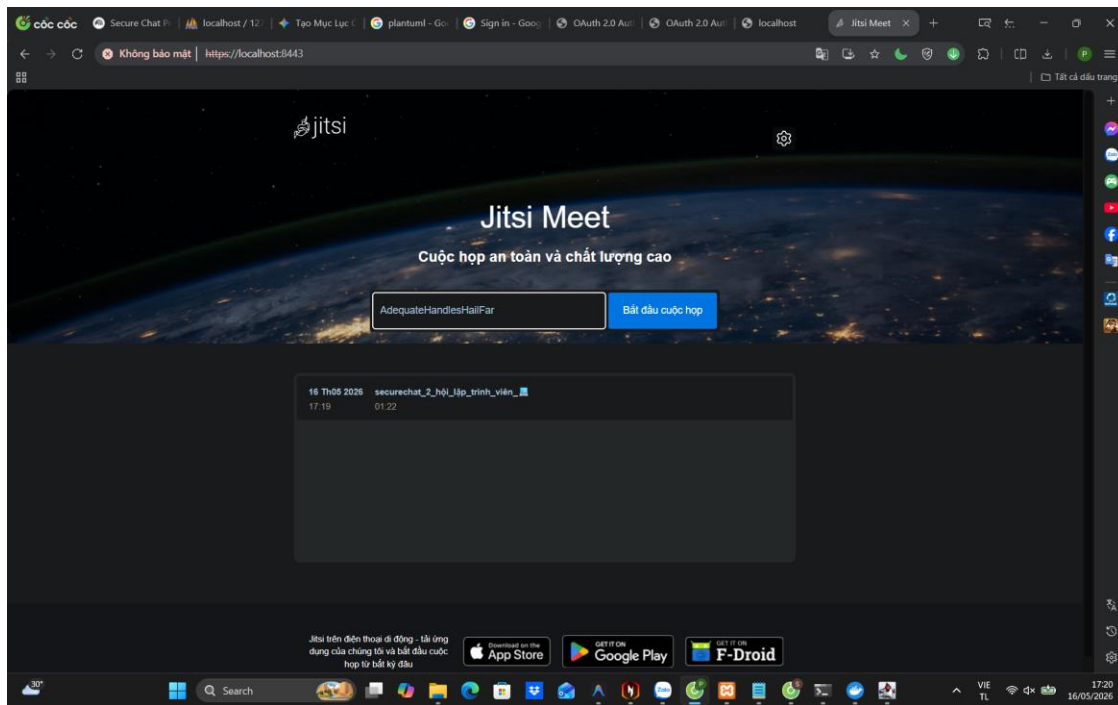
Mô tả UI: Tin nhắn cũ mờ đi, ô nhập liệu hiện lên. Tin sau khi sửa có nhãn “(Đã chỉnh sửa)” đảm bảo tính minh bạch.



Hình 4.21: Giao diện Trợ lý Lập lịch / Nhắc việc

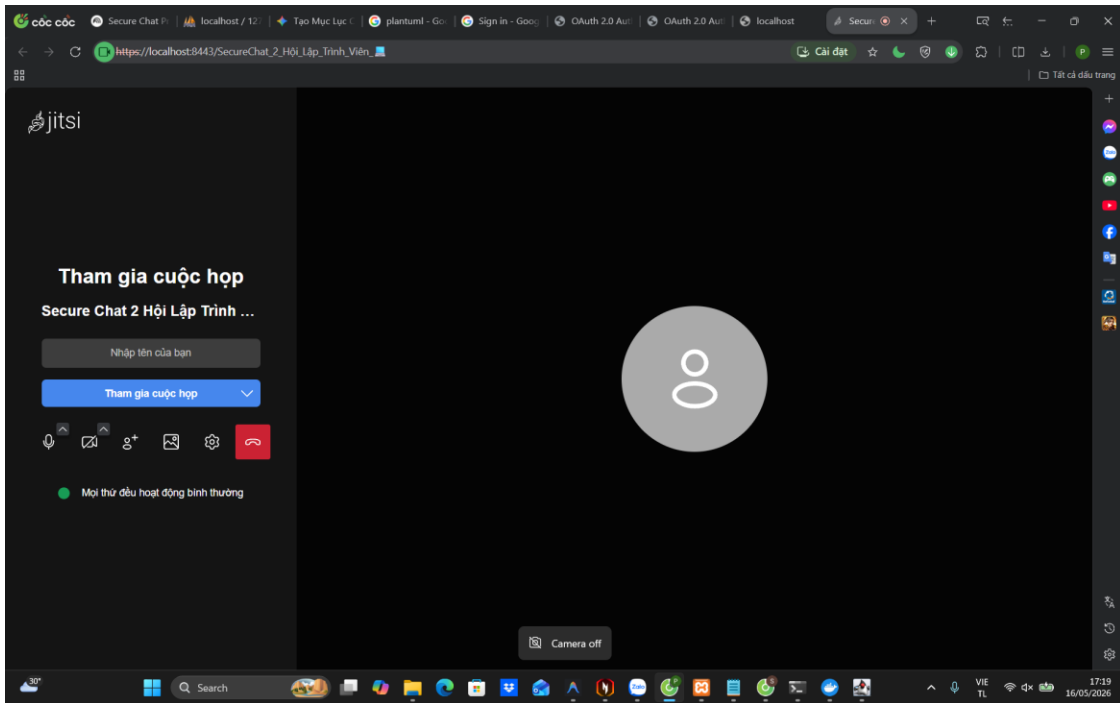
Mô tả UI: To-do List tích hợp trong chat. Có checkbox, Deadline và nhãn độ ưu tiên (Priority labels).

4.5. Phân hệ Giao tiếp Đa phương tiện (Jitsi Video Call)

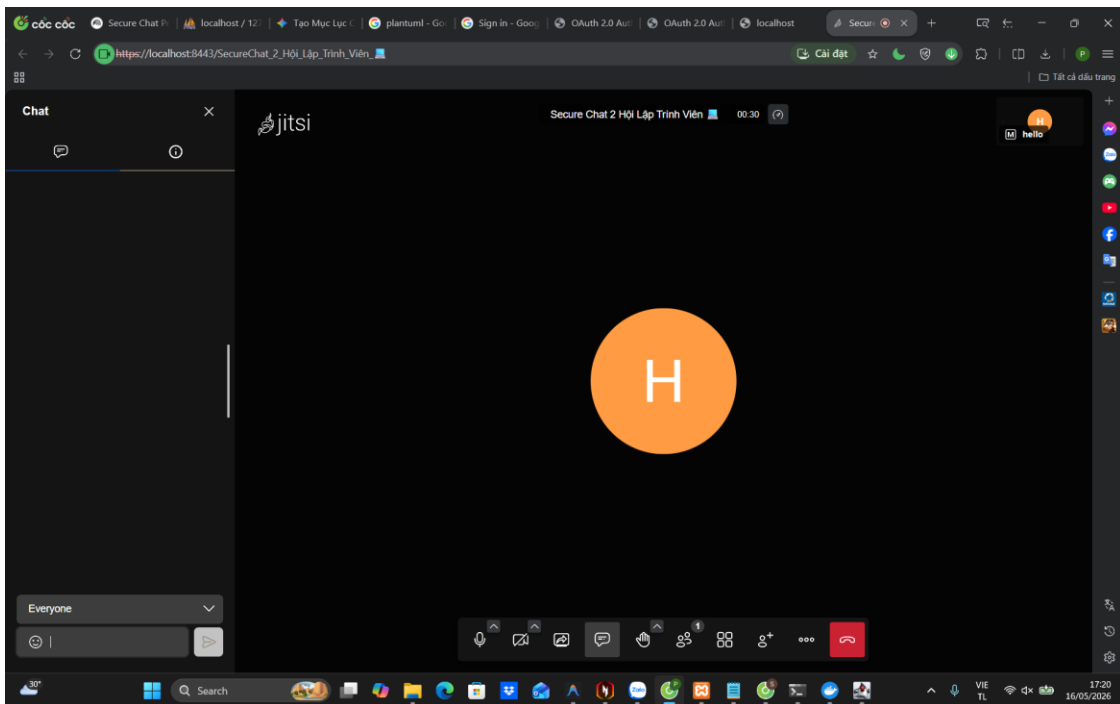


Hình 4.22: Giao diện khởi tạo phòng họp Video Call

Mô tả UI: Giao diện Dark mode. Có ô nhập tên phòng và lịch sử các cuộc gọi gần đây.

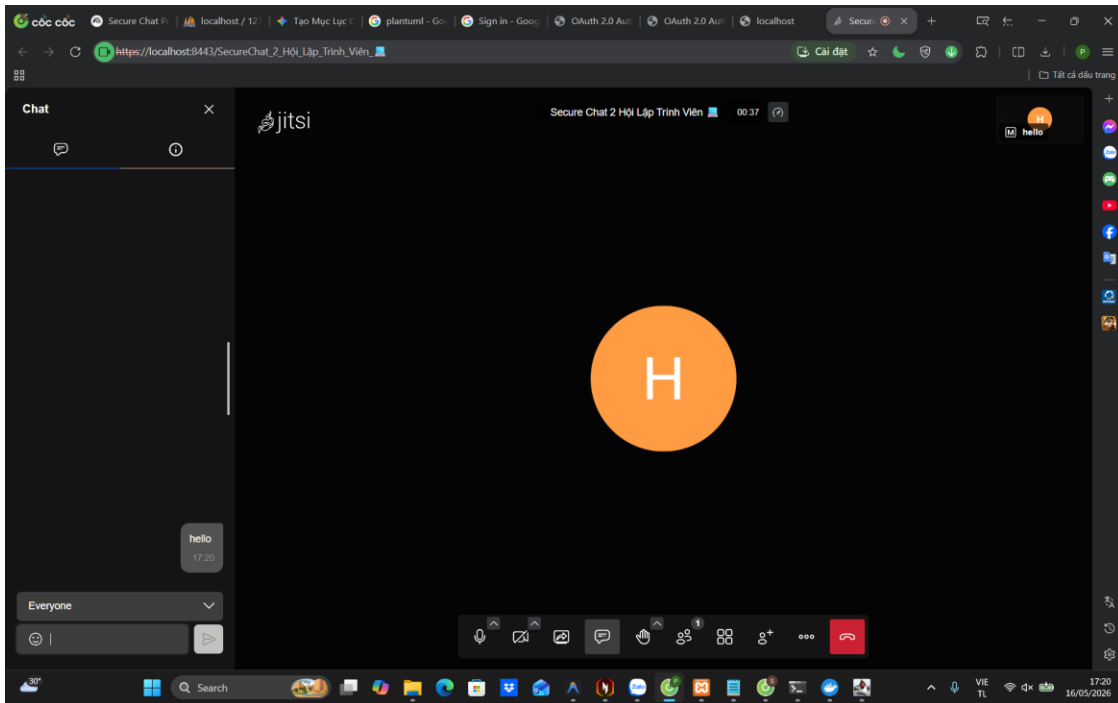


Hình 4.23: Màn hình Phòng chờ (Lobby / Pre-join)



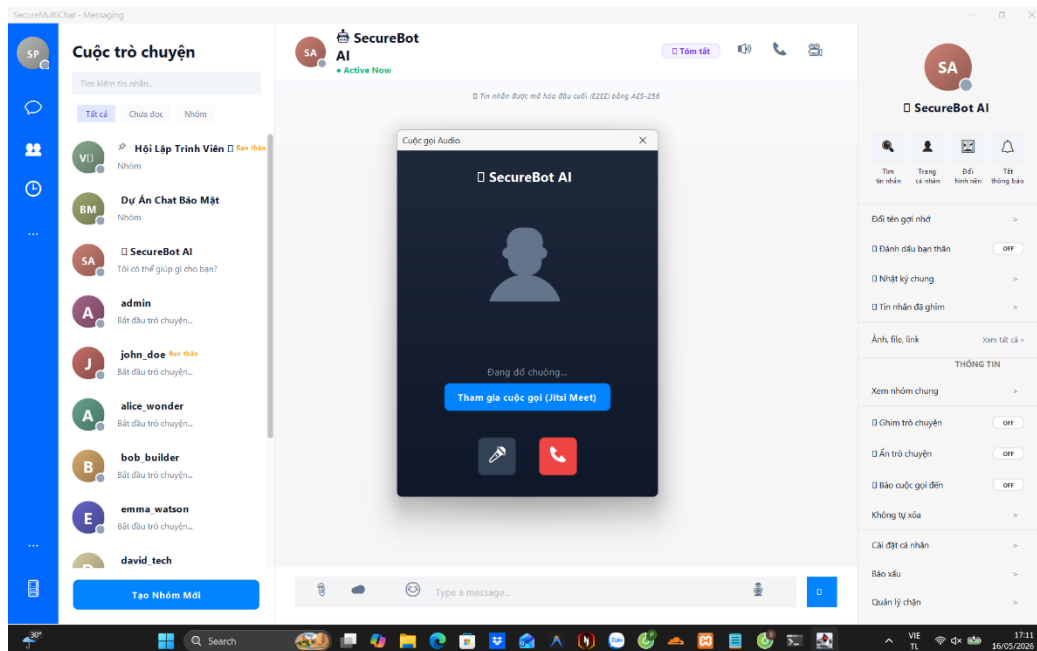
Hình 4.24: Giao diện Cuộc gọi Video (Trạng thái thu gọn)

Mô tả UI: Không gian hiển thị (Viewport) mở rộng tối đa. Nút điều khiển bo tròn trên nền đen mờ, không làm phân tâm.



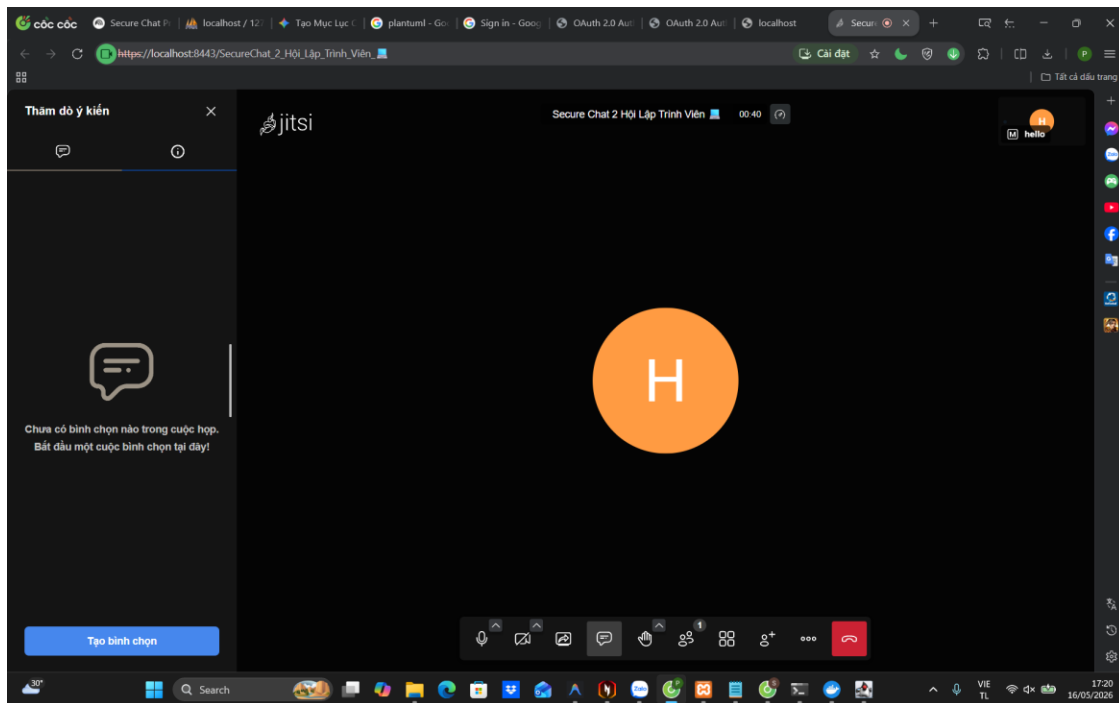
Hình 4.25: Giao diện Cuộc gọi Video & Khung Chat

Mô tả UI: Toolbar chứa đầy đủ công cụ: Share màn hình, Giờ tay, Tile view. Khung chat song song bên trái.



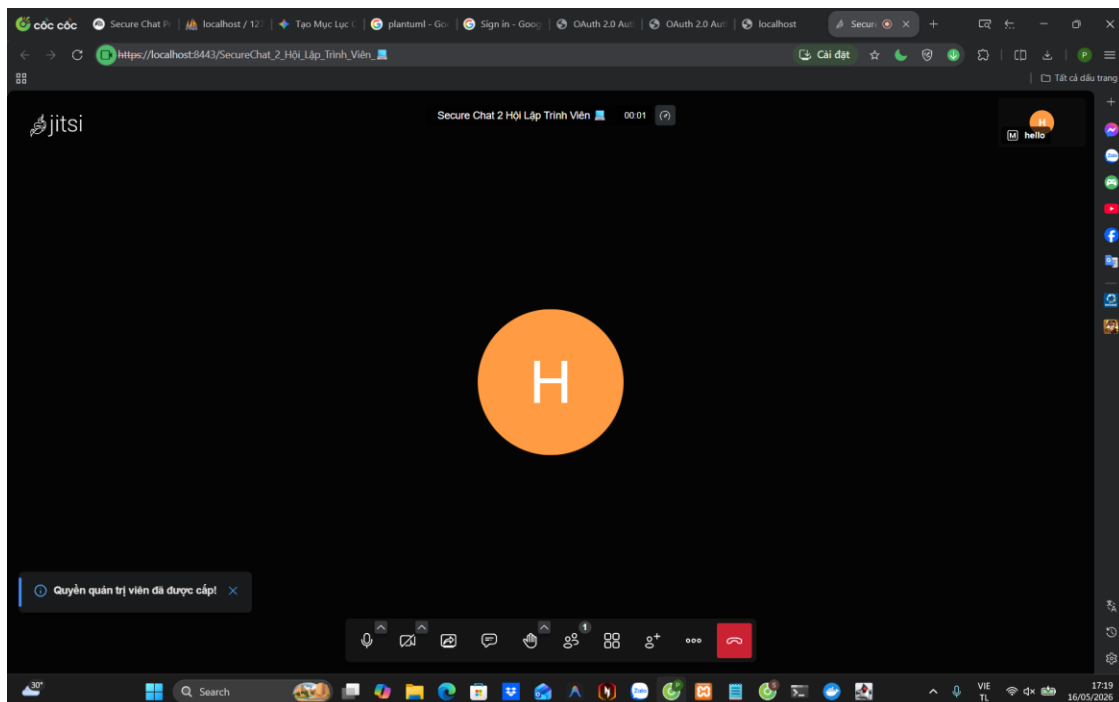
Hình 4.26: Giao diện Cuộc gọi Âm thanh (Audio Call)

Mô tả UI: Tối giản với Avatar giữa màn hình. Hiệu ứng gợn sóng âm thanh. Toolbar chỉ giữ 3 nút thiết yếu.



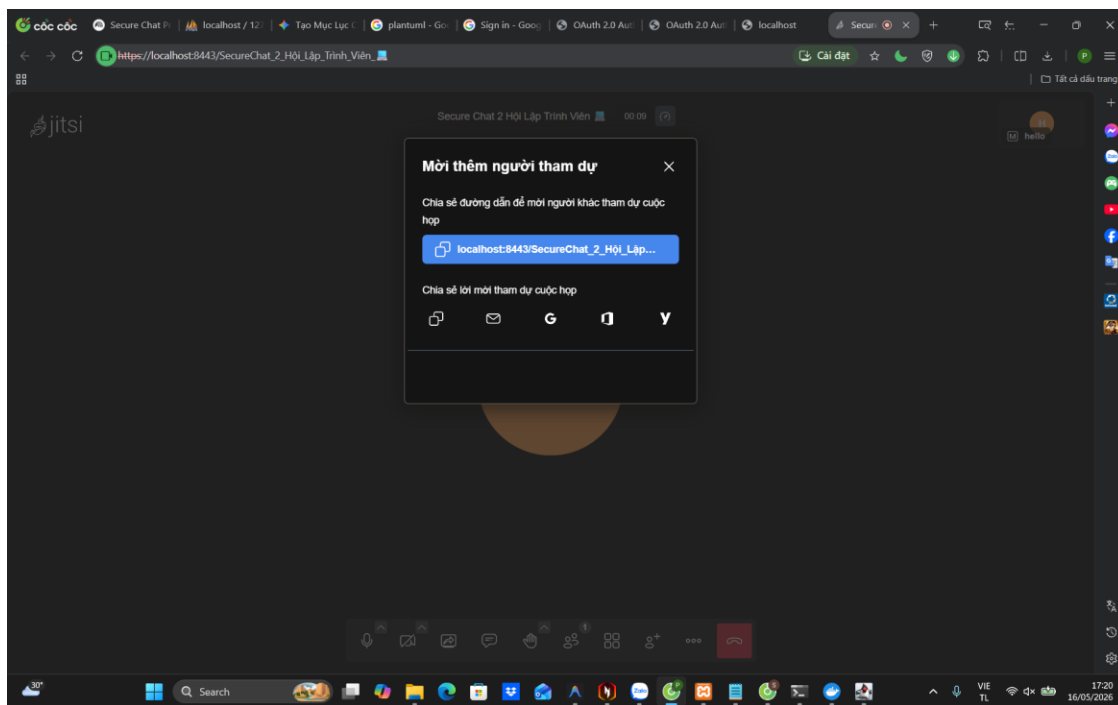
Hình 4.27: Giao diện tạo Thăm dò ý kiến (Poll)

Mô tả UI: Tab tương tác cho phép tạo bình chọn trực tiếp ngay trong cuộc gọi.



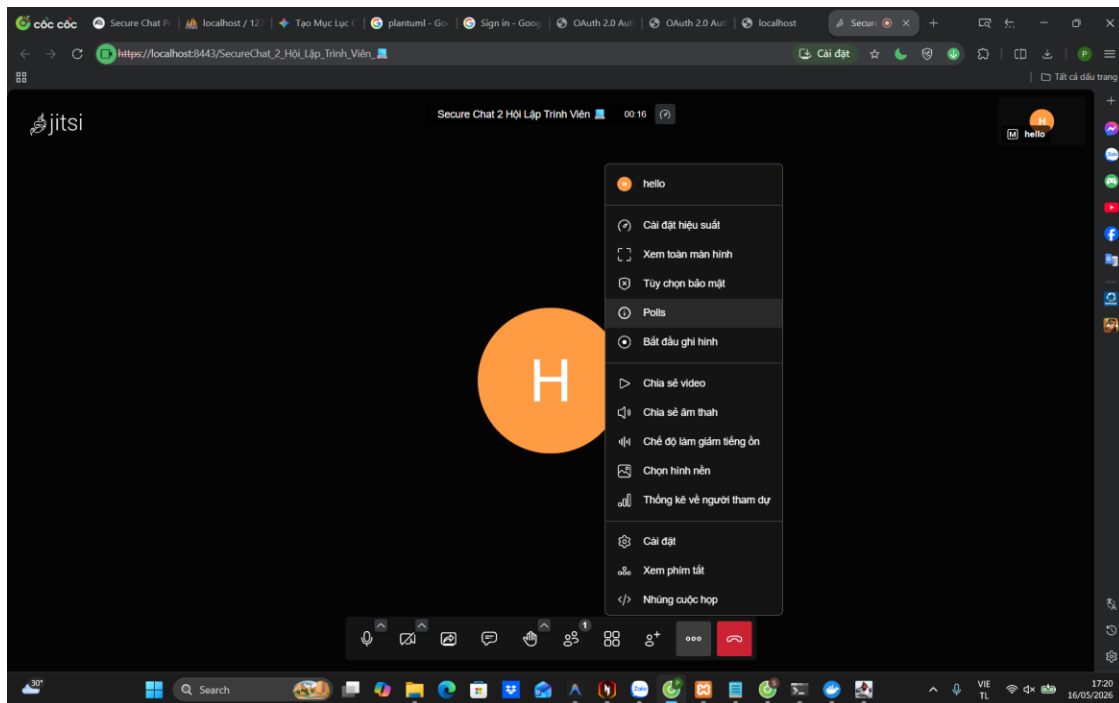
Hình 4.28: Thông báo phân quyền Quản trị viên

Mô tả UI: Toast notification góc dưới báo hiệu hệ thống RBAC tự động gán quyền Host.



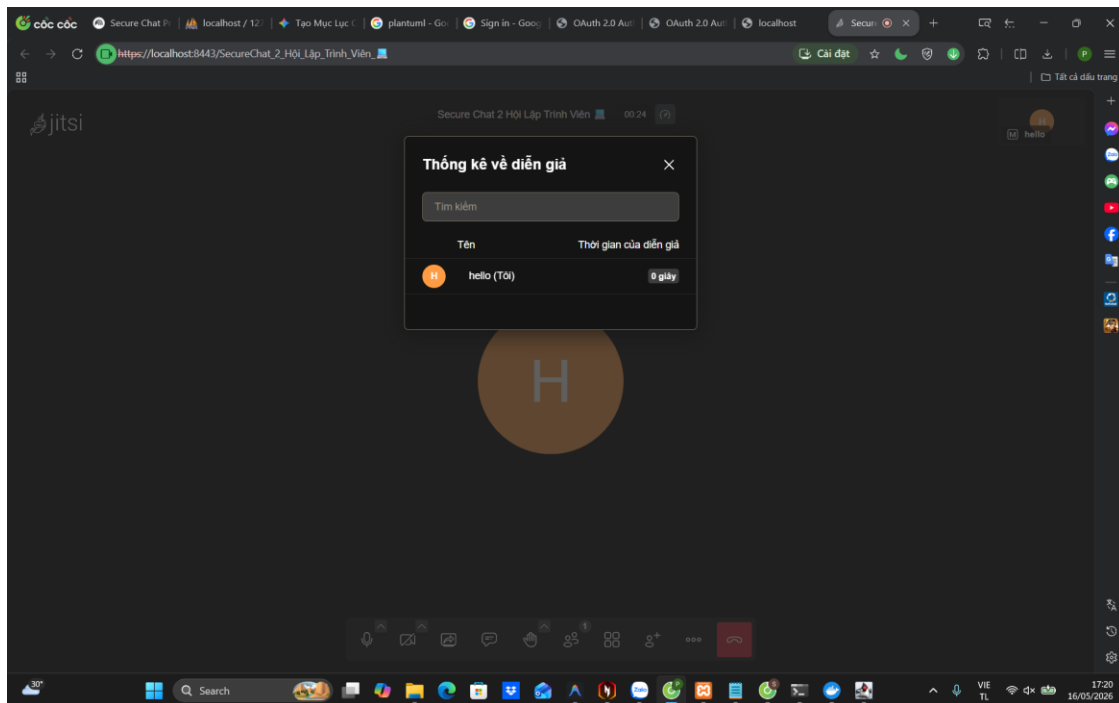
Hình 4.29: Giao diện Mời người tham dự

Mô tả UI: Modal tối giản có nút Copy một chạm. Có Deep links tới Google, Outlook để chia sẻ nhanh.



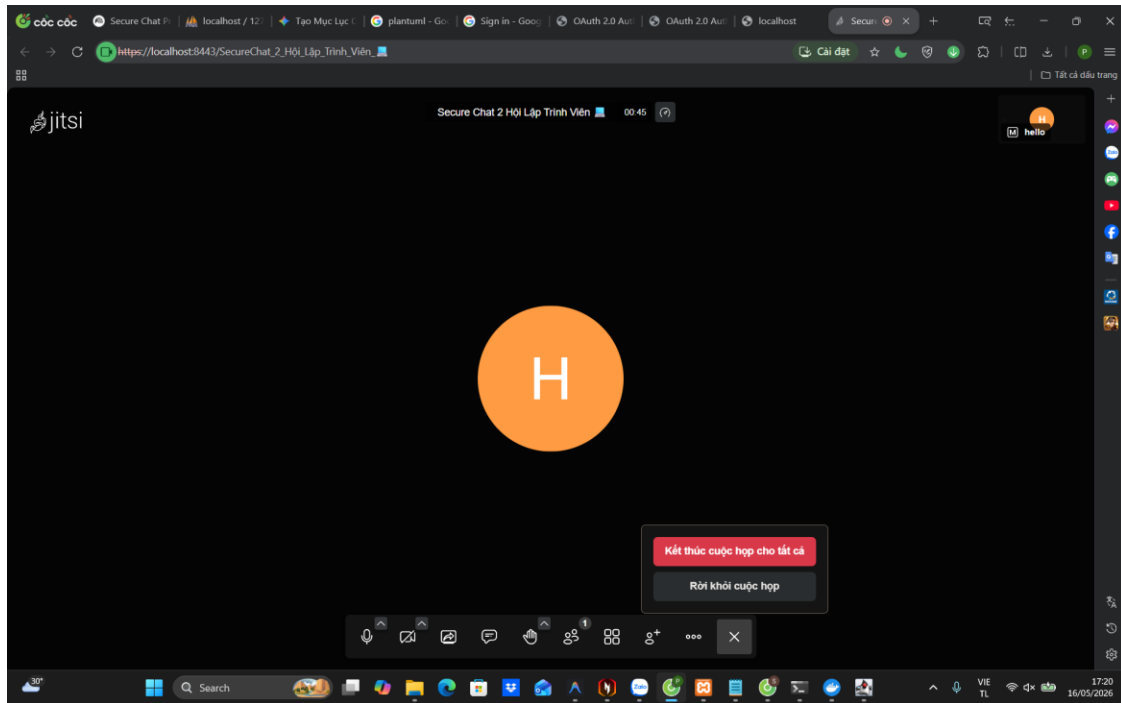
Hình 4.30: Menu Tùy chọn nâng cao (Context Menu)

Mô tả UI: Menu dọc chứa Cài đặt hiệu suất, Ghi hình, Chọn nền ảo. Giải phóng không gian cho Toolbar chính.



Hình 4.31: Giao diện Thống kê diễn giả (Speaker Stats)

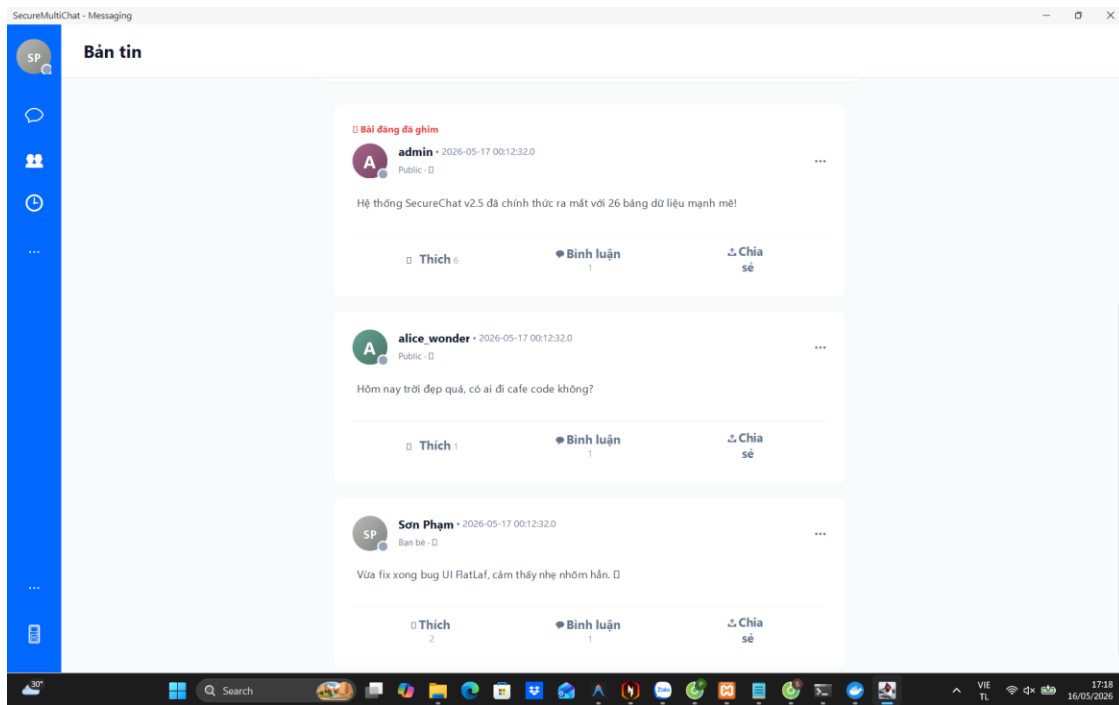
Mô tả UI: Data table hiển thị danh sách người tham gia kèm bộ đếm thời gian phát biểu thực tế.



Hình 4.32: Giao diện xác nhận kết thúc cuộc gọi

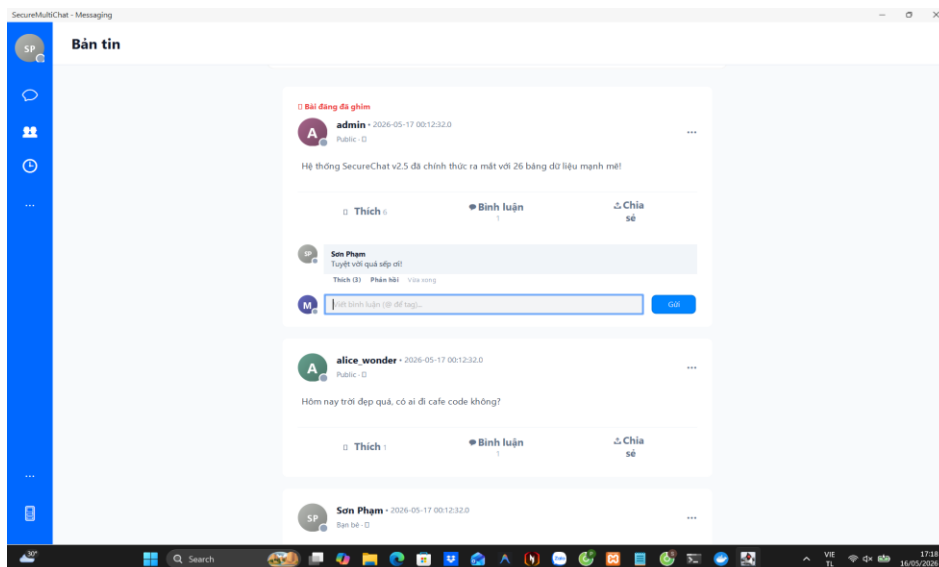
Mô tả UI: Menu Popup phân quyền: “Kết thúc cho tất cả” (Host) hoặc “Rời khỏi” (Thành viên), tránh thoát nhầm.

4.6. Phân hệ Mạng xã hội nội bộ và Cộng tác



Hình 4.33: Tổng quan Trang Bản tin (Newsfeed)

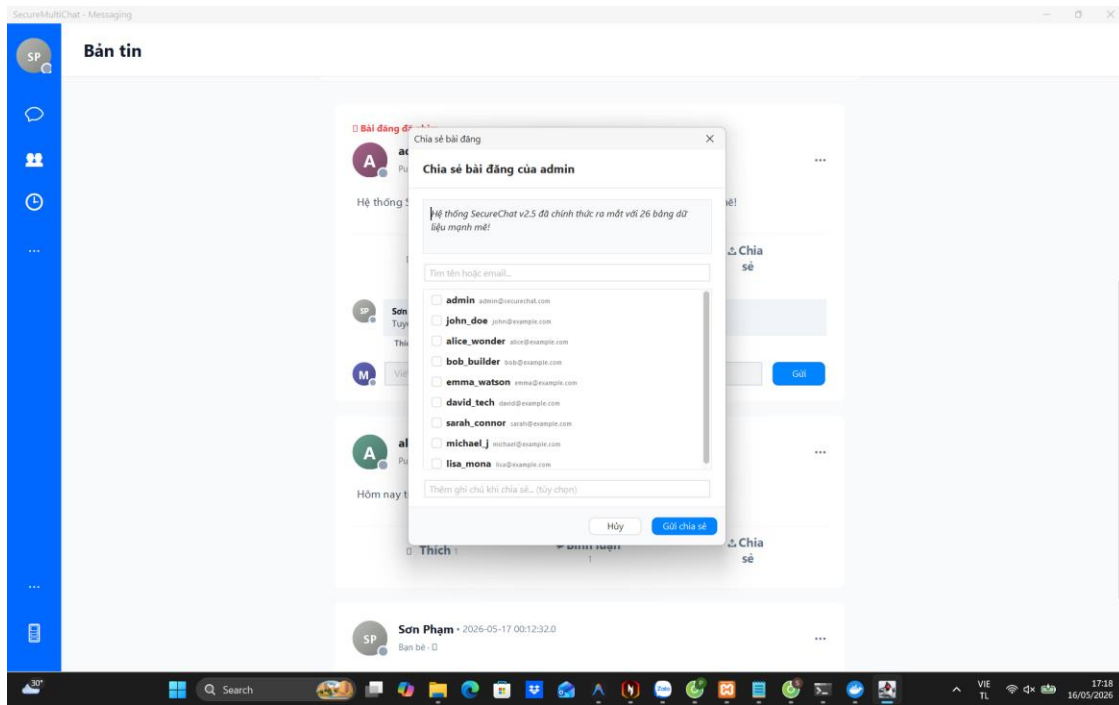
Mô tả UI: Thiết kế Card UI trên nền off-white. Hỗ trợ “Bài đăng đã ghim”. Nút tương tác (Thích, Bình luận, Share) có bộ đếm.



Hình 4.34: Giao diện tương tác Bình luận

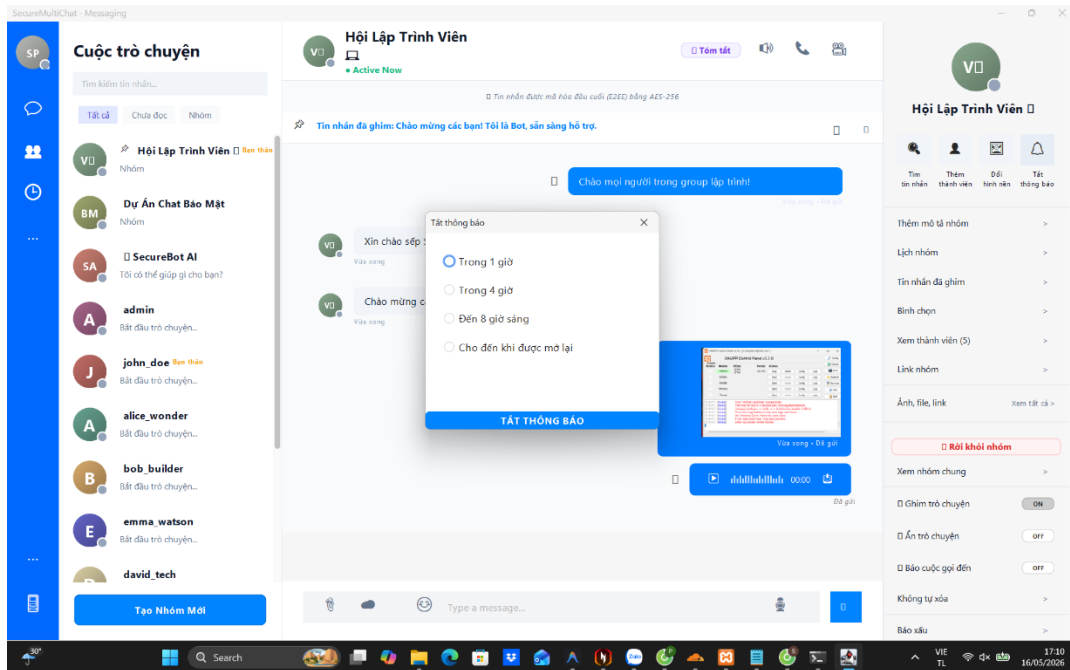
Mô tả UI: Khu vực bình luận thụt lề (indent). Hỗ trợ Placeholder “@ để tag” nhắc tên người dùng.

Hình 4.47: Cửa sổ Popup Chia sẻ bài đăng



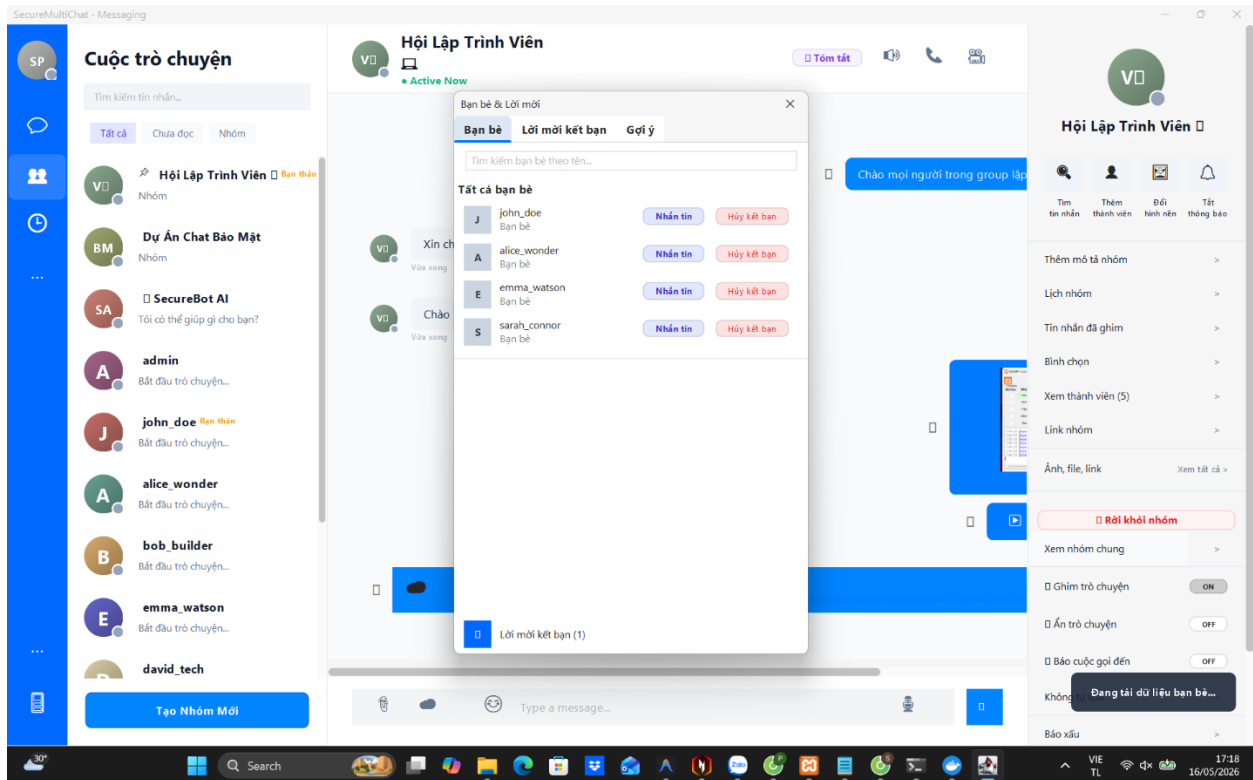
Hình 4.35: Cửa sổ Popup Chia sẻ bài đăng

Mô tả UI: Modal Overlay. Danh sách bạn bè dạng Checkbox (Multi-select) để chia sẻ định tuyến.



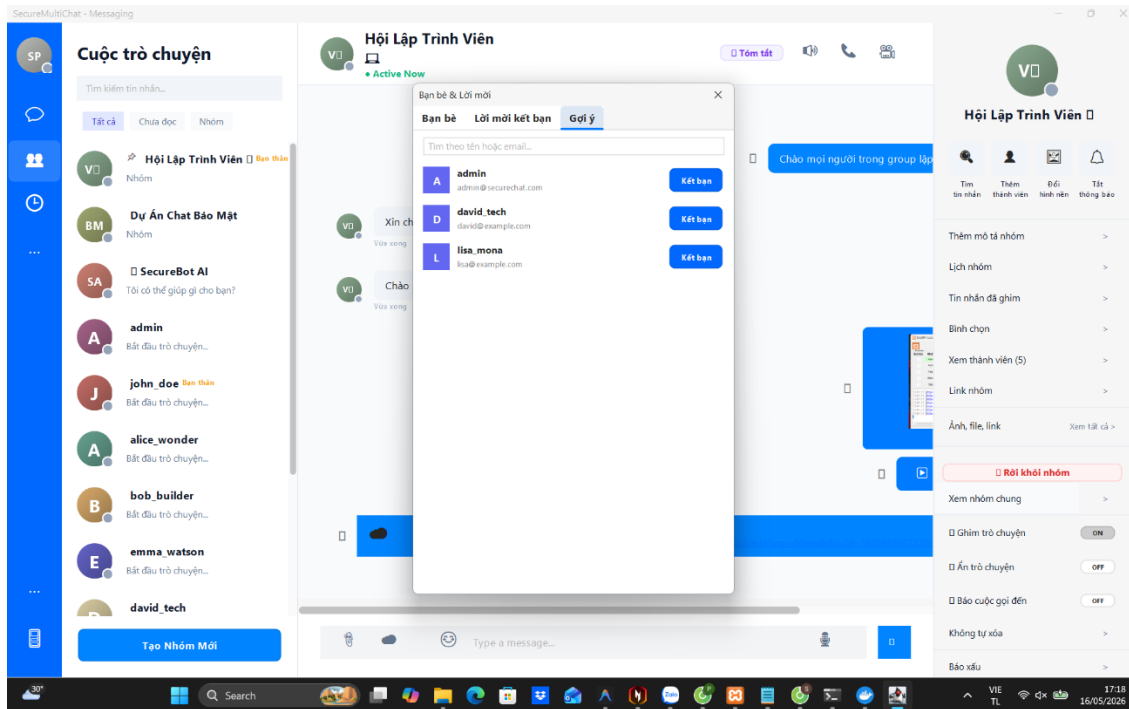
Hình 4.36: Bảng điều khiển Thông báo (Notification)

Mô tả UI: Dropdown Menu từ Quả chuông. Thông báo chưa đọc được Highlight và có Red Badge (Dấu chấm đỏ).



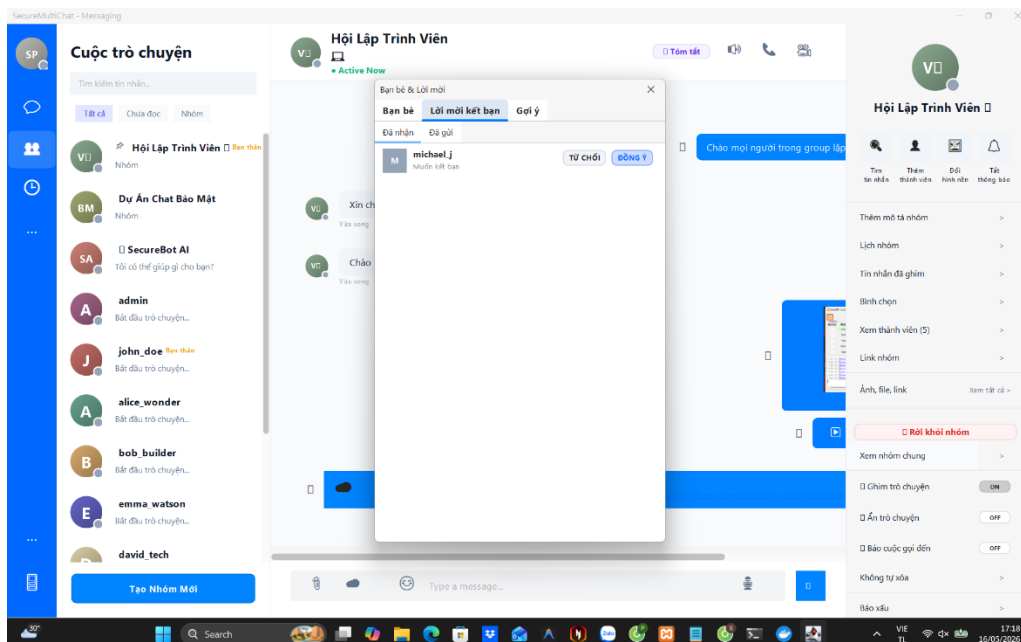
Hình 4.37: Cửa sổ Popup Gợi ý Kết bạn

Mô tả UI: Tabbed Navigation (Bạn bè, Lời mời, Gợi ý). Nút CTA “Kết bạn” nổi bật màu xanh dương.



Hình 4.38: Giao diện Gửi Lời mời Kết bạn

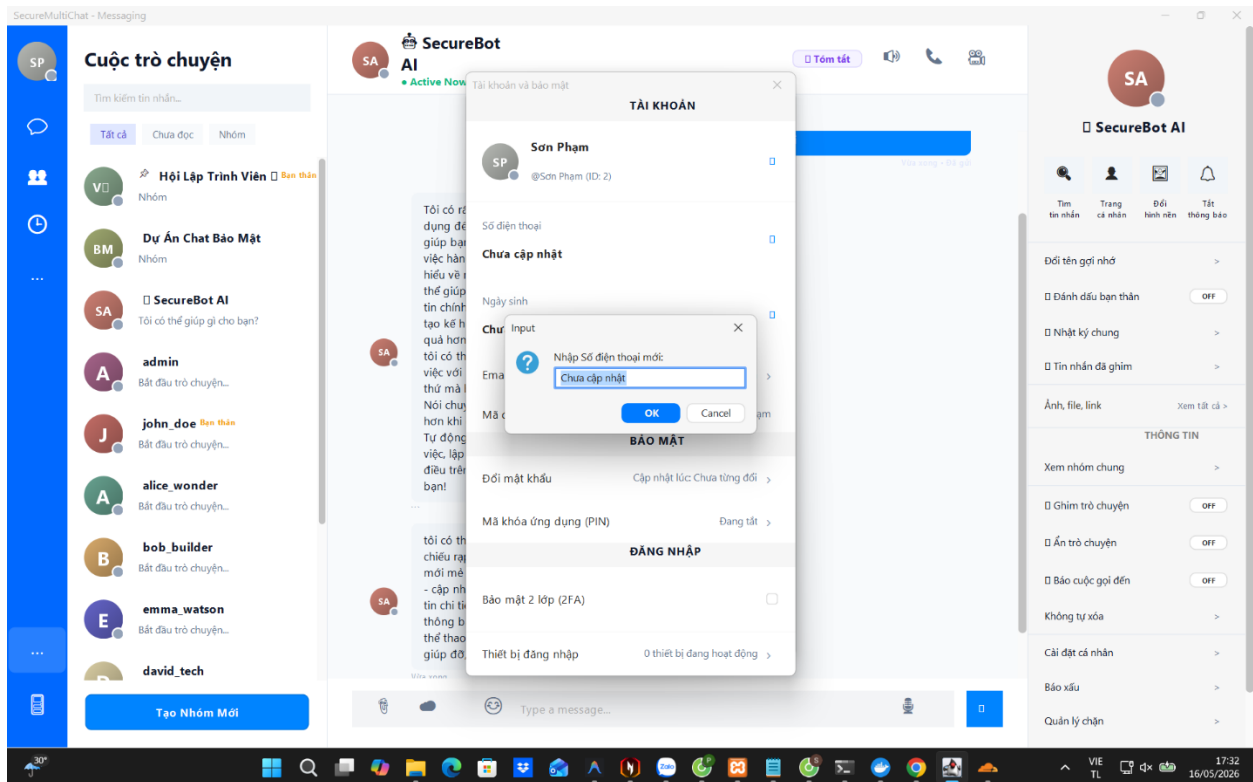
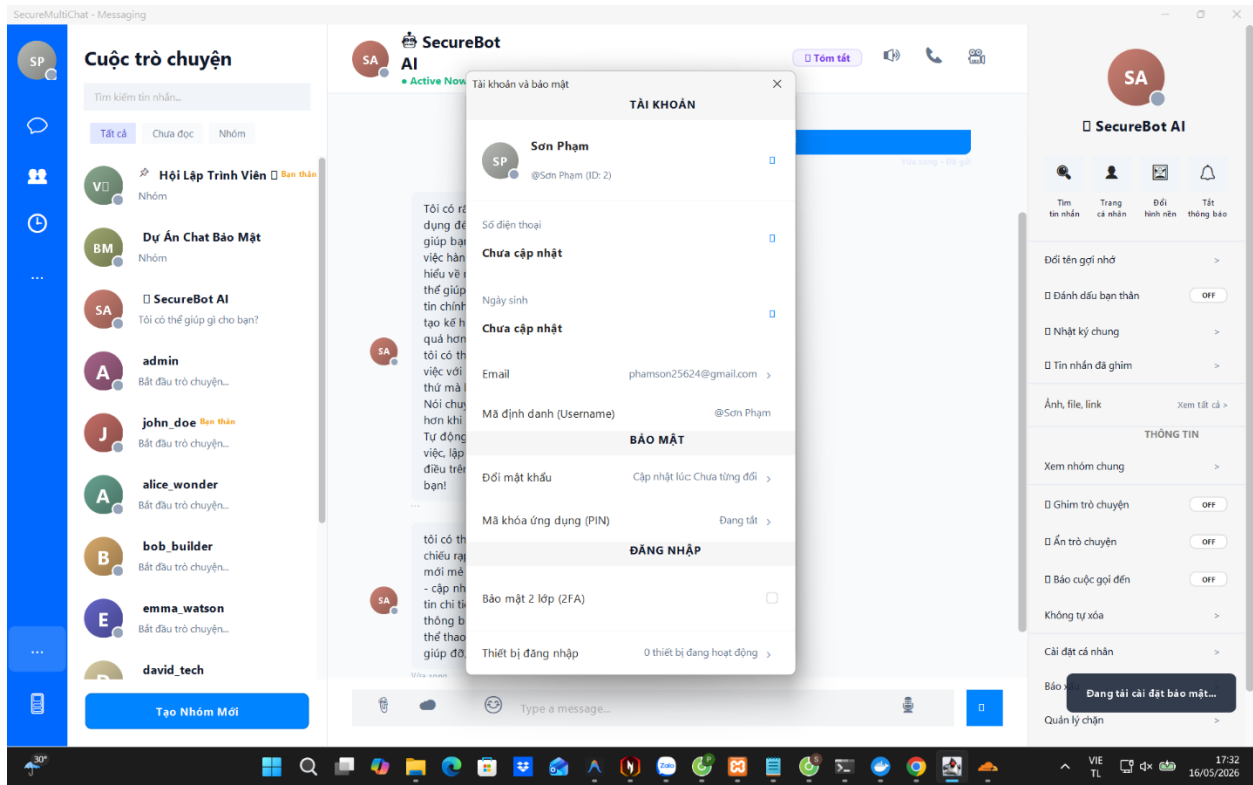
Mô tả UI: Nút CTA chuyển sang “Đã gửi” và bị vô hiệu hóa (Disabled) sau khi bấm để chống Spam.



Hình 4.39: Danh sách Lời mời Kết bạn

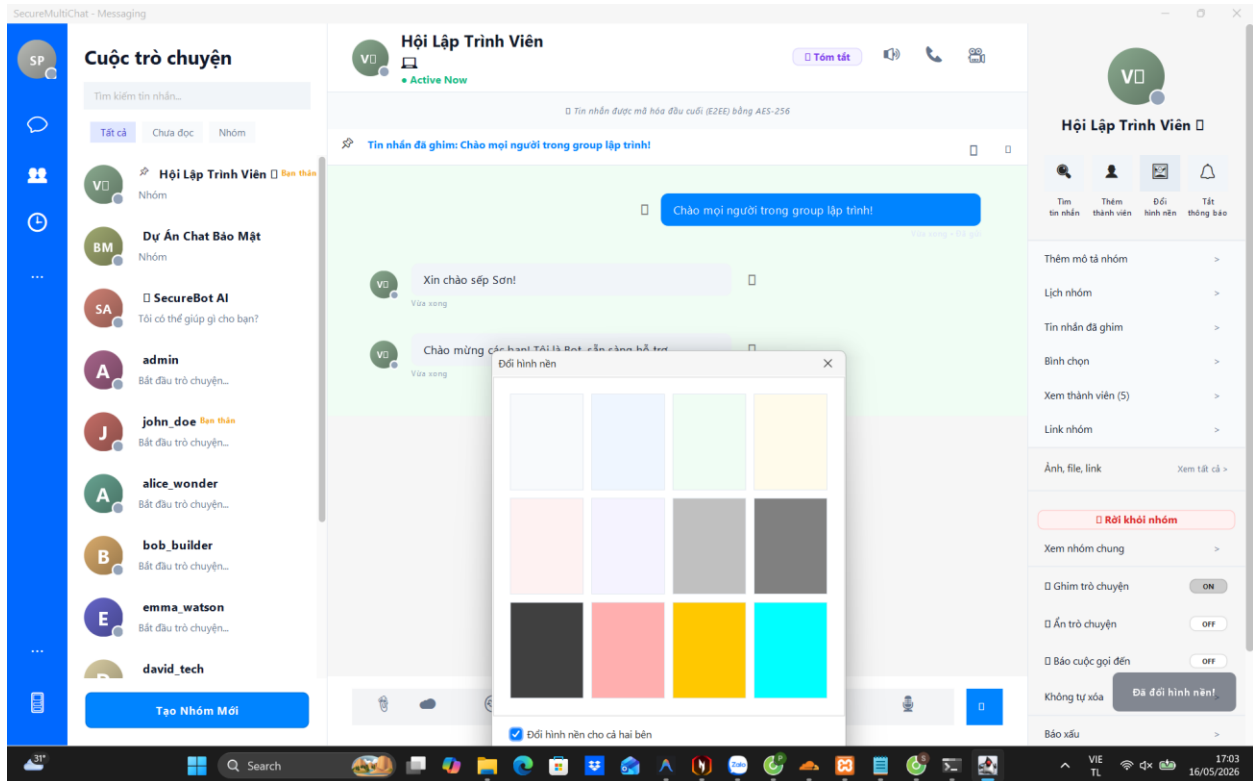
Mô tả UI: Thẻ hiển thị 2 nút rõ ràng: Xanh lá (Chấp nhận) và Đỏ (Từ chối) - tuân thủ Clear Affordance.

4.7. Phân hệ Cài đặt và Tùy chỉnh (Settings & Configs)



Hình 4.40: Cài đặt Thông tin Cá nhân

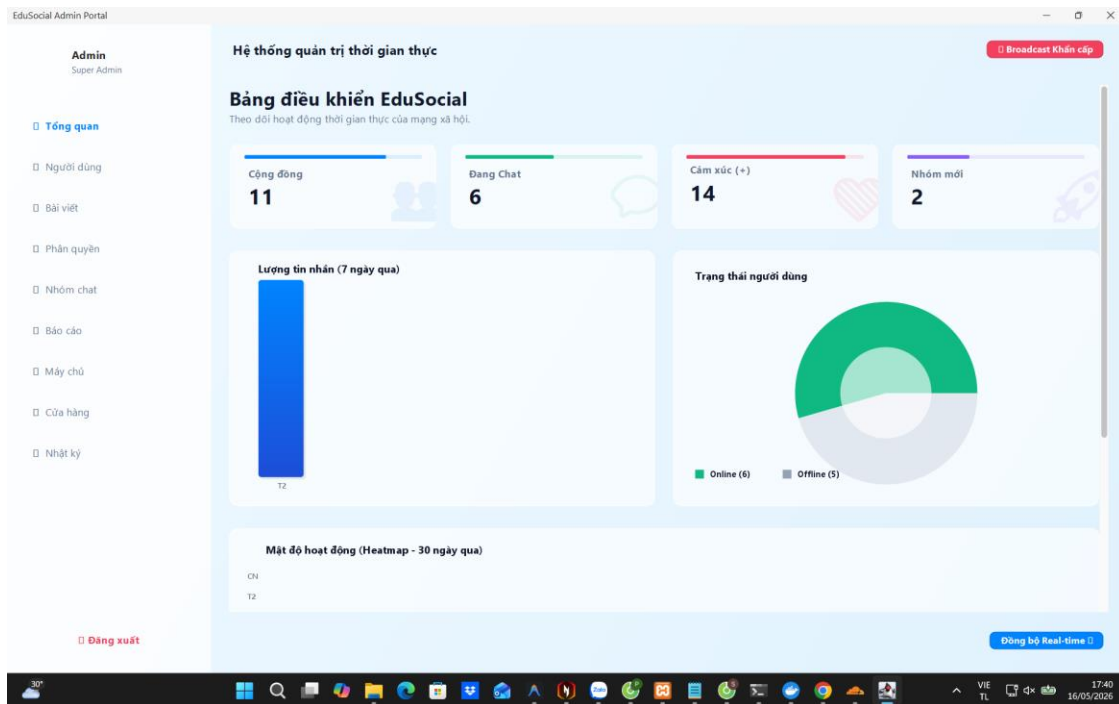
Mô tả UI: Avatar có Camera overlay icon. Nút “Cập nhật” bị khóa (Disabled) nếu không có sự thay đổi liệu.



Hình 4.41: Đổi Hình nền Cuộc trò chuyện

Mô tả UI: Cung cấp Tab phân loại (Màu trơn, Thư viện, Upload). Có khung Preview trước khi “Áp dụng”.

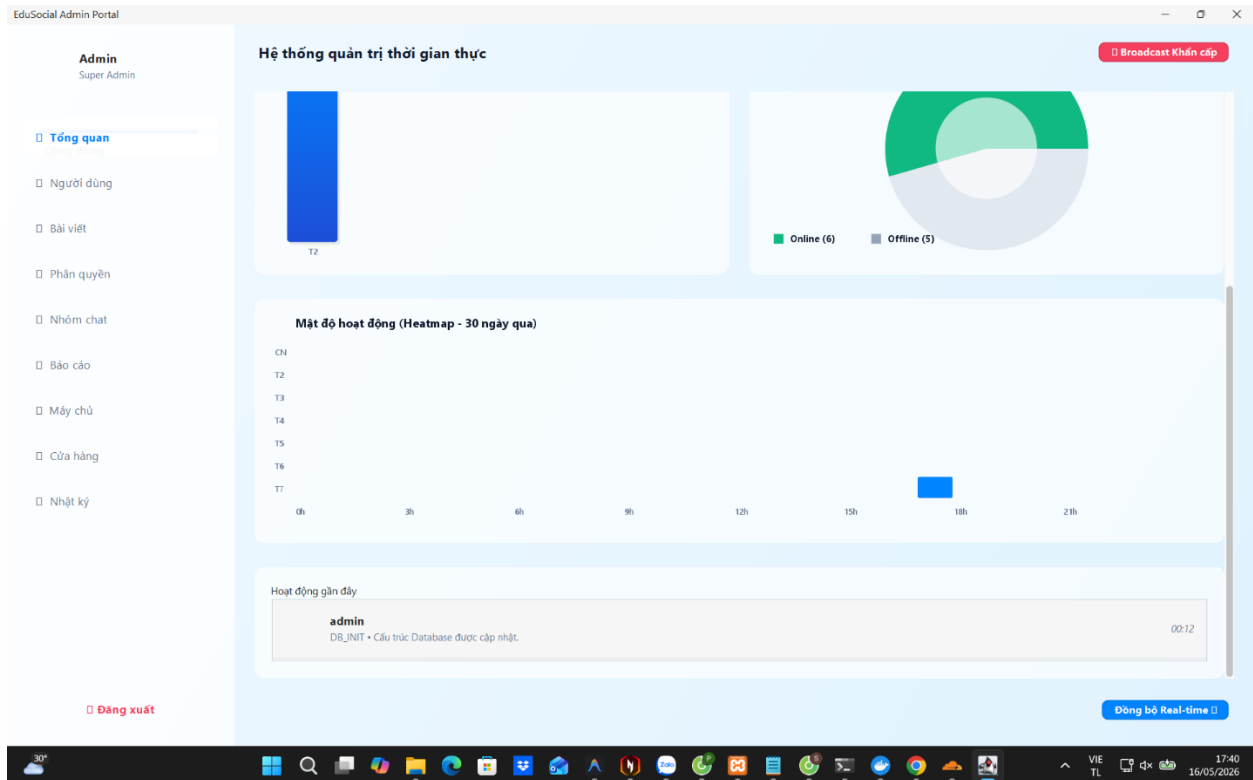
4.8. Phân hệ Quản trị Admin và Bảo mật Hệ thống



Hình 4.52: Tổng quan Bảng điều khiển Quản trị (Admin Dashboard)

Mô tả UI: Sidebar cố định. Hiển thị KPIs bằng Stat Cards (Tổng user, RAM/CPU) và Biểu đồ trực quan.

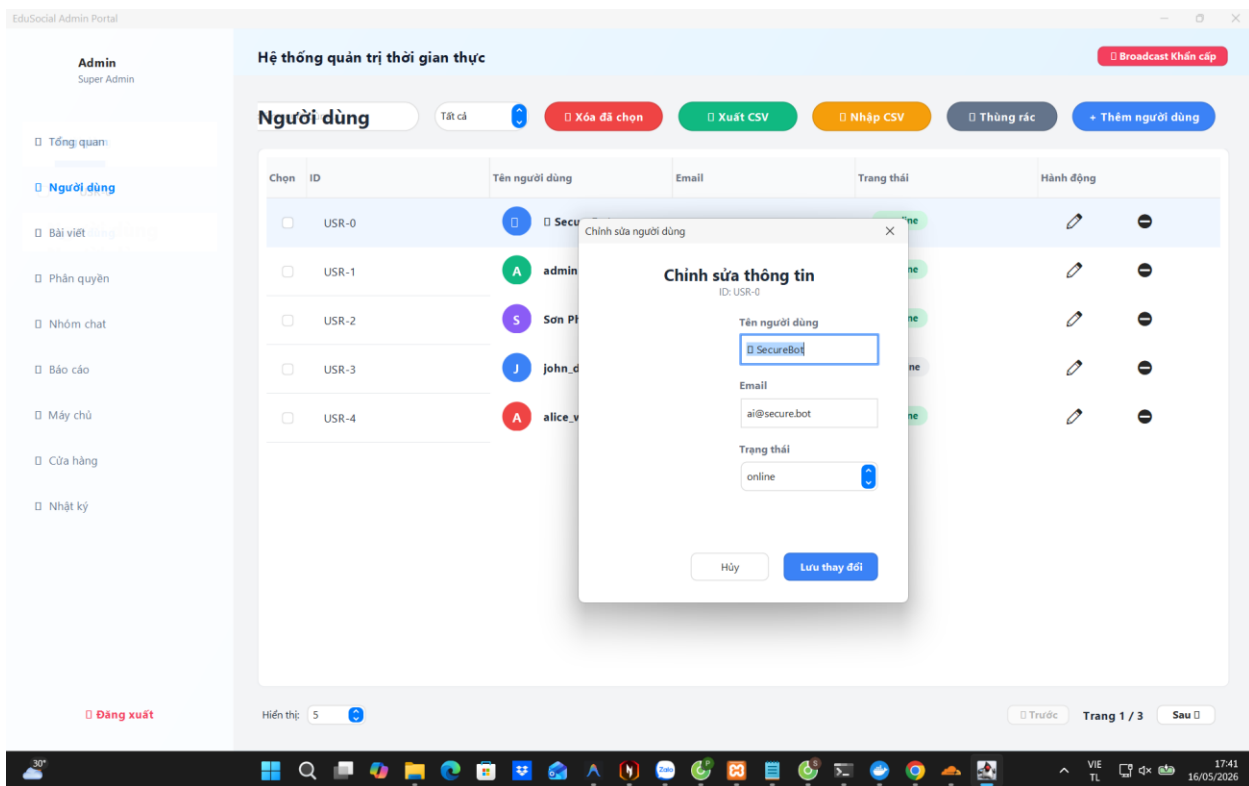
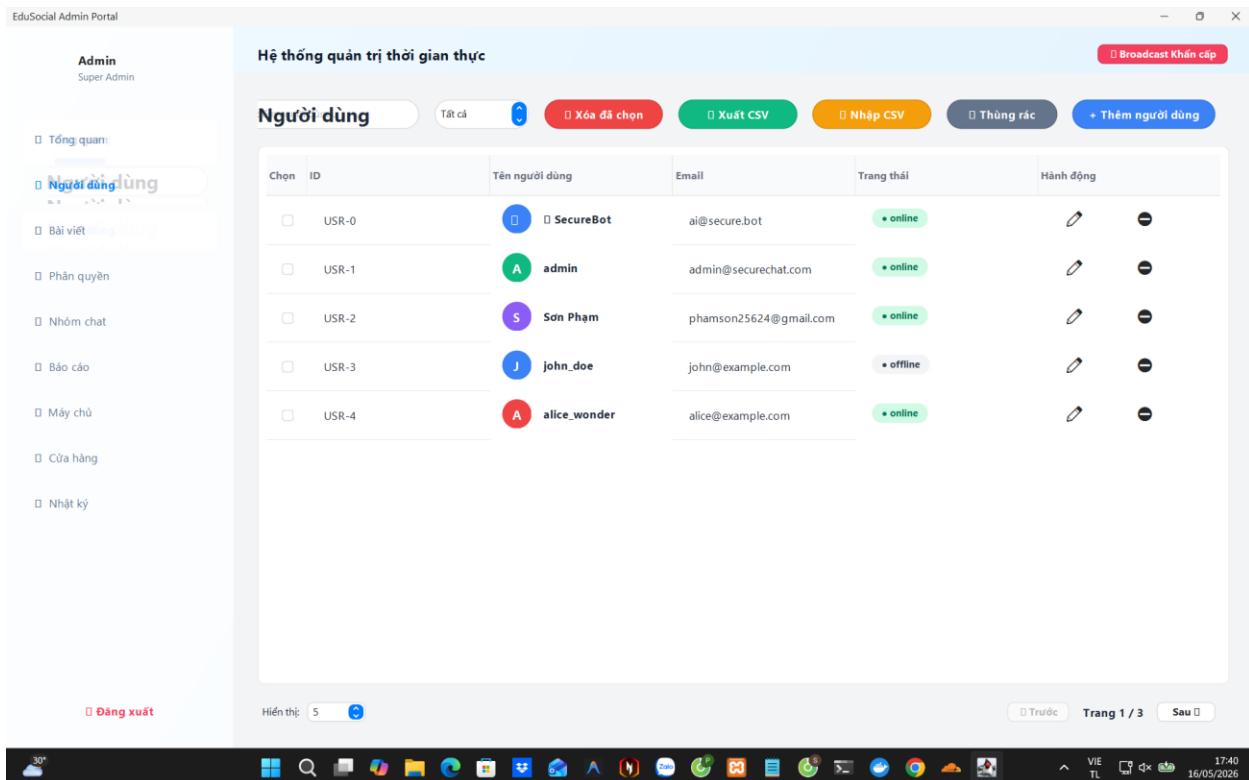
Hình 4.72: Bảng điều khiển Thống kê hoạt động

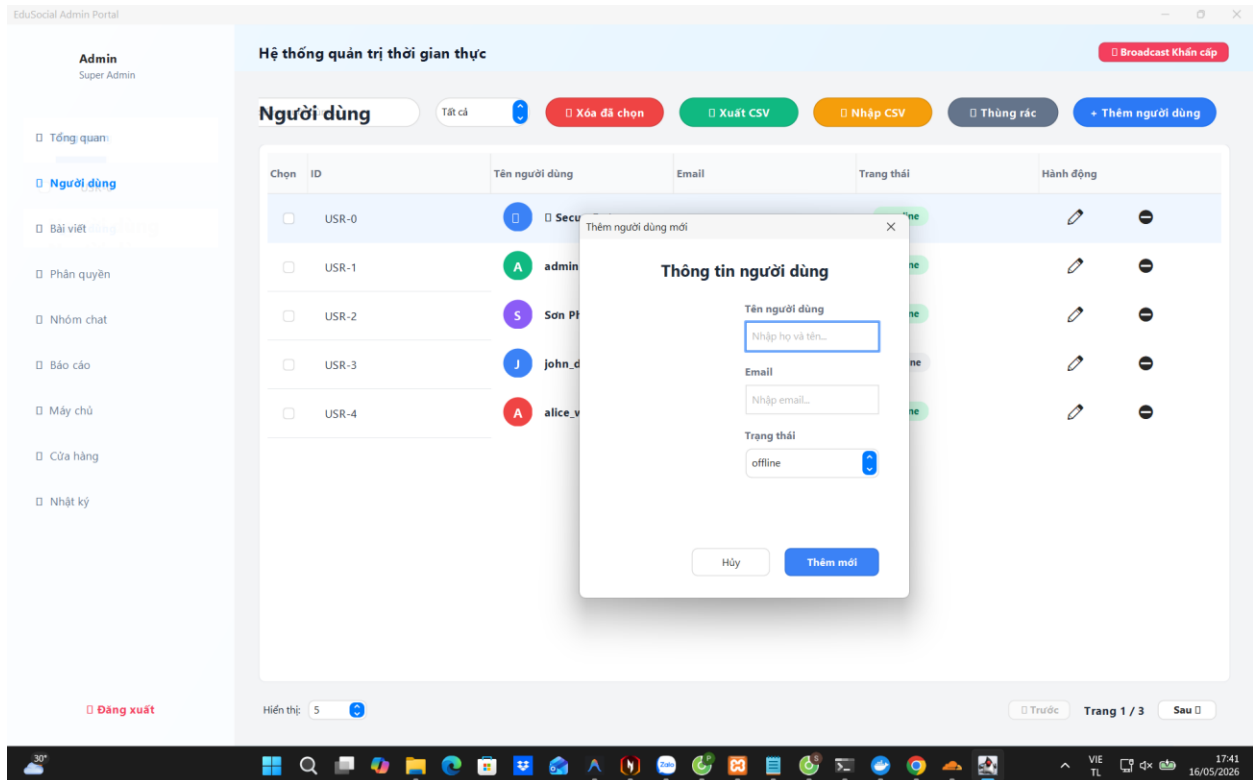


Hình 4.72: Bảng điều khiển Thống kê hoạt động

Mô tả UI: Top Bar chứa Summary Cards với Sparklines. Main Area dùng Bar/Pie chart phân tích xu hướng data.

Hình 4.73: Giao diện Quản lý Người dùng (Data Table)

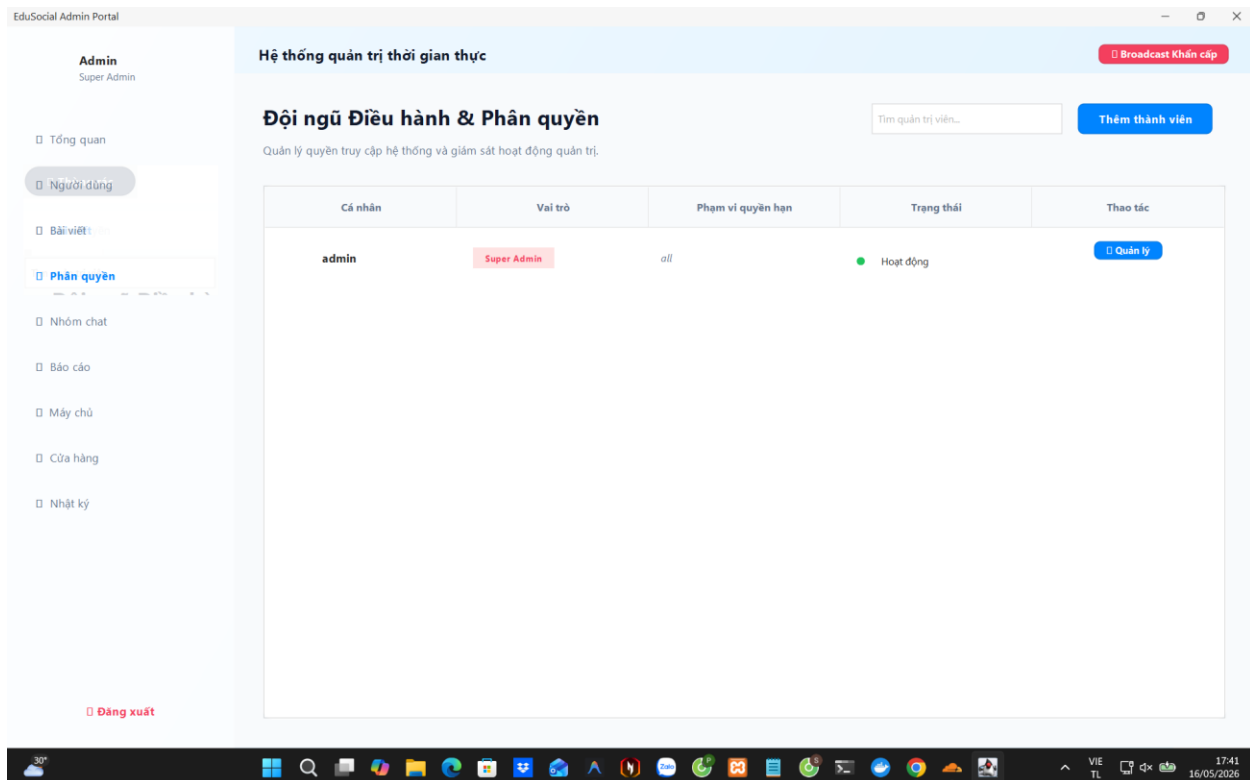




Hình 4.73: Giao diện Quản lý Người dùng (Data Table)

Mô tả UI: Bảng tích hợp Thanh tìm kiếm, Bộ lọc và Phân trang (Pagination). Cột Actions thao tác khóa/mở tài khoản.

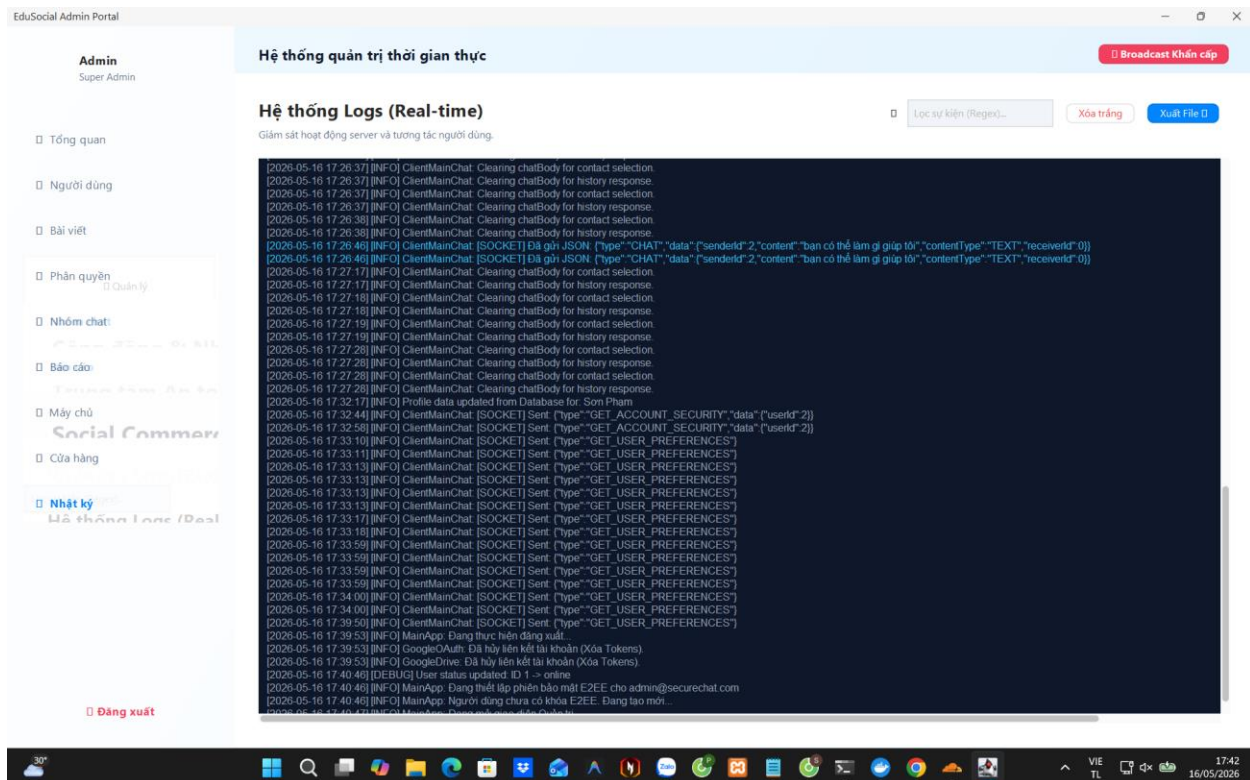
Hình 4.74: Cửa sổ Phân quyền Người dùng (RBAC)



Hình 4.74: Cửa sổ Phân quyền Người dùng (RBAC)

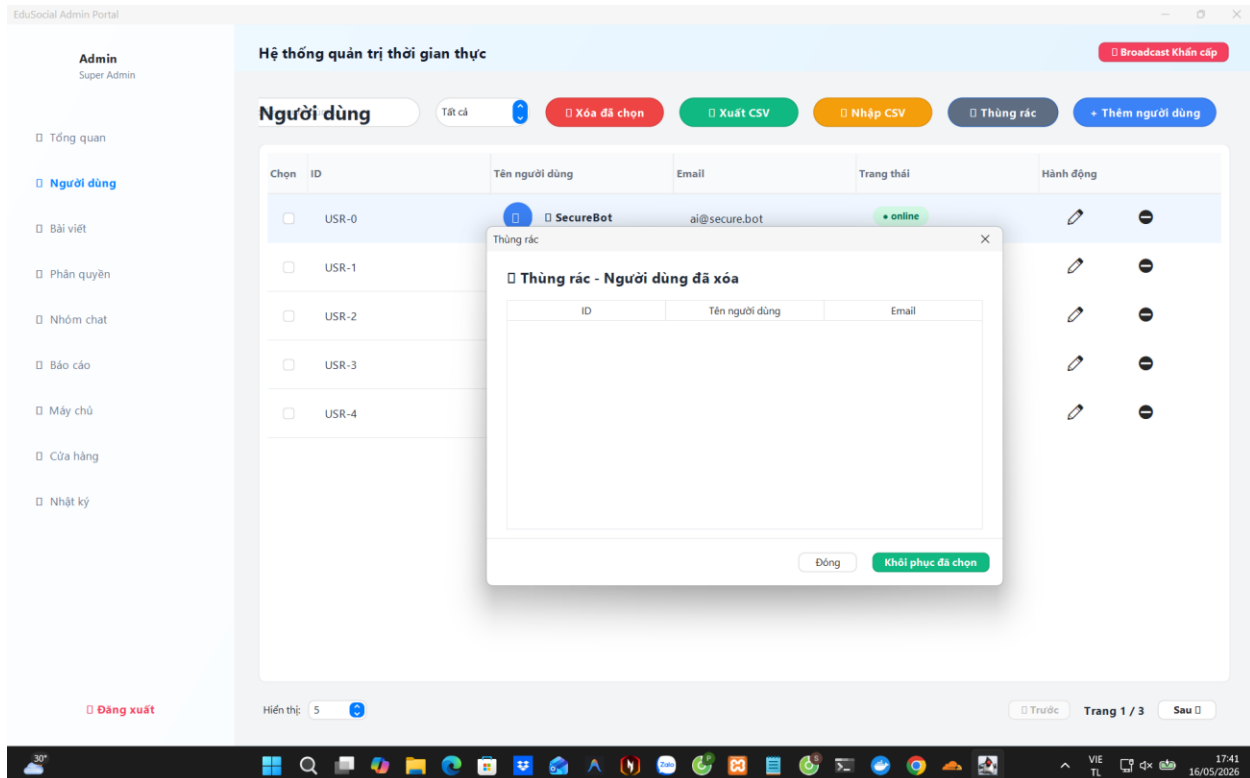
Mô tả UI: Ma trận Checkbox (Matrix checkboxes) gán quyền chi tiết (Xem, Sửa, Xóa) cho từng Module.

Hình 4.75: Nhật ký Hoạt động (System Audit Logs)



Hình 4.75: Nhật ký Hoạt động (System Audit Logs)

Mô tả UI: Danh sách log (Timestamp, Action, Actor, Status) thiết kế dạng text đề cao tính dễ đọc và truy vết.



Hình 4.80: Danh sách Người dùng bị chặn

Mô tả UI: List View kèm nút “Bỏ chặn” (Unblock). Có Empty State thân thiện nếu danh sách trống.

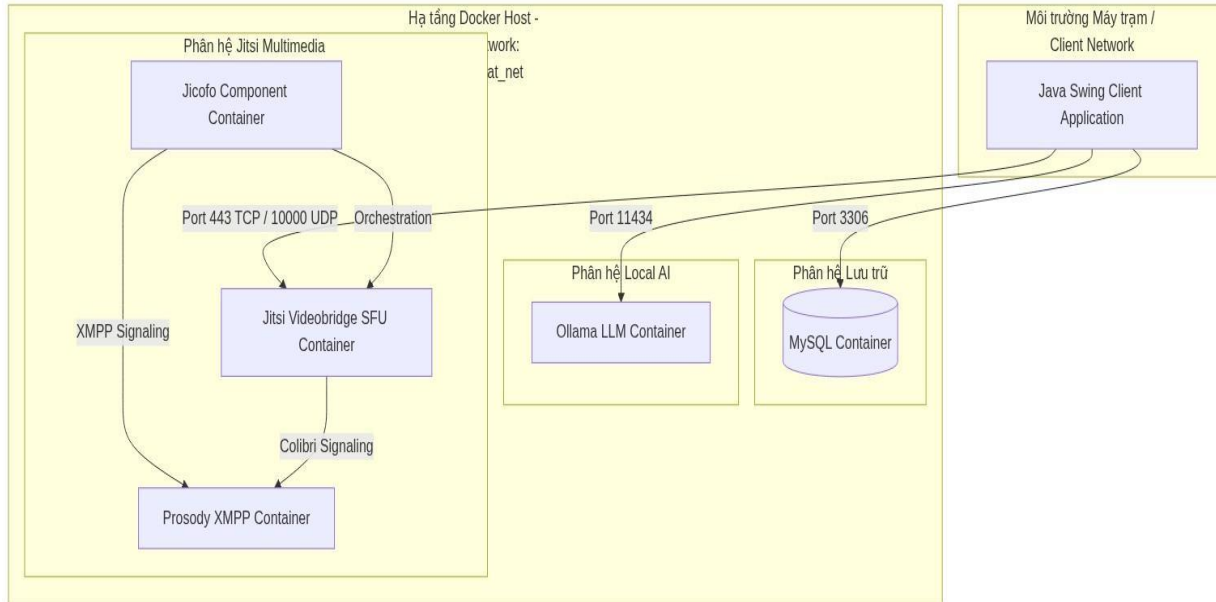
CHƯƠNG 5: CÀI ĐẶT VÀ KIỂM THỬ (IMPLEMENTATION & TESTING)

5.1.1. Thiết lập file docker-compose.yml (Giải thích các service: ollama, mysql, jitsi)

Hạ tầng tự vận hành (Self-hosted Infrastructure) của hệ thống Secure Chat được đóng gói và quản lý tập trung thông qua công nghệ Container hóa của Docker, cụ thể là công cụ orchestration docker-compose. Việc thiết lập này thiết lập một môi trường phân tán cô lập, cho phép định nghĩa, liên kết cấu trúc liên mạng và cấu hình tài nguyên phần cứng đồng bộ cho toàn bộ hệ thống back-end. Cấu trúc tệp tin docker-compose.yml được thiết kế để phân rã hệ thống thành các phân hệ độc lập, giao tiếp với nhau qua cấu trúc mạng nội bộ (Bridge Network) an toàn.

1. Cấu trúc kiến trúc phân tầng mạng và Phân rã Dịch vụ (Services Decomposition)

Hạ tầng hệ thống được chia làm ba cụm dịch vụ cốt lõi tương ứng với ba vai trò kỹ thuật: Lưu trữ trạng thái bền vững (MySQL), Xử lý suy luận ngôn ngữ cục bộ (Ollama), và Truyền thông đa phương tiện thời gian thực (Jitsi Meet Stack). Sơ đồ dưới đây mô tả kiến trúc liên kết mạng và mô hình phân bổ cổng (Port Mapping) của các container trong hệ thống:



Hình 26: Kiến trúc liên mạng và ánh xạ cổng dịch vụ trong cấu hình Docker Compose.

2. Giải thích chi tiết các Service cấu hình học thuật

Cơ chế này đảm bảo tính toàn vẹn và bất biến của dữ liệu (Data Persistence) ngay cả khi Container bị hủy hoặc khởi động lại. Thuộc tính **restart: always** được cấu hình để tự động phục hồi dịch vụ khi xảy ra lỗi crash hệ thống.

Dịch vụ ollama (Tầng suy luận Trí tuệ nhân tạo cục bộ):

Vai trò kỹ thuật: Đóng gói framework Ollama để thực thi tiến trình suy luận (Inference) của các mô hình ngôn ngữ lớn (Llama 3 / Mistral) hoàn toàn offline. Service này cung cấp một cổng giao tiếp API cục bộ dạng RESTful tại cổng **11434**, cho phép ứng dụng Client tương tác trực tiếp mà không cần cấu hình phức tạp.

Phân bổ tài nguyên phần cứng (Hardware Acceleration): Điểm cốt lõi mang tính học thuật trong việc cấu hình ollama trong Docker là việc thiết lập thẻ **deploy.resources.reservations.devices**. Cấu hình này kích hoạt driver **nvidia** cho phép Container truy cập trực tiếp vào nhân CUDA của GPU vật lý thông qua kỹ thuật **passthrough**. Việc tối ưu hóa phần cứng giúp chuyển đổi quá trình tính toán ma trận trọng số từ CPU sang GPU, giảm thiểu thời gian tạo Token đầu tiên (TTFT) xuống mức dưới giây.

Cụm dịch vụ Jitsi Meet Stack (prosody, jicofo, jvb):

Vai trò kỹ thuật: Phân rã cấu trúc liên hợp của Jitsi thành ba container riêng biệt để tối ưu hóa hiệu năng định tuyến:

1. **Prosody** đóng vai trò là máy chủ báo hiệu XMPP, quản lý trạng thái hiện diện và phòng họp ảo (MUC).
2. **Jicofo** (Jitsi Conference Focus) đóng vai trò điều phối phiên, chịu trách nhiệm phân bổ các kênh truyền thông đa phương tiện.

3. **JVB** (Jitsi Videobridge) là thực thể SFU (Selective Forwarding Unit) chịu trách nhiệm định tuyến trực tiếp các luồng video/audio RTP/SRTP giữa các Client mà không thực hiện giải mã, giúp giảm tải CPU máy chủ đáng kể.

Cấu hình mạng và Ràng buộc giao thức: Khác với các dịch vụ thông thường, jvb yêu cầu mở dải cổng UDP diện rộng (mặc định là cổng **10000**) thông qua chế độ ánh xạ trực tiếp để phục vụ cho giao thức thiết lập kết nối WebRTC (ICE Candidates). Các biến môi trường (environment) như **XMPP_SERVER**, **JICOFO_AUTH_USER**, và **JVB_STUN_SERVERS** được cấu hình chéo để thiết lập mối quan hệ tin cậy và đồng bộ hóa trạng thái bắt tay truyền thông giữa ba thành phần này.

3. Bản đặc tả tham số cấu hình tệp tin docker-compose.yml

Ma trận đặc tả tham số kỹ thuật các Service trong cấu hình Docker Compose

Tên Service	Hình ảnh Gốc (Docker Image)	Ánh xạ Cổng (Port Mapping)	Biến môi trường cốt lõi	Cấu hình Tài nguyên & Lưu trữ
mysql	mysql:8.0	3306:3306 (TCP)	MYSQL_ROOT_PASSWORD=secure_password MYSQL_DATABASE=secure_chat_db	Volume: mysql_data:/var/lib/mysql
ollama	ollama/ollama:latest	11434:11434 (TCP)	OLLAMA_NUM_PARALLEL=4	capabilities: [gpu] (NVIDIA Runtime) Volume: ollama_storage:/root/.ollama
prosody	jitsi/prosody:stable	Mạng nội bộ (Internal)	XMPP_DOMAIN=auth.securechat.local GLOBAL_MODULES=bosh,pubsub,ping	Signaling backbone (XMPP)
jicofo	jitsi/jicofo:stable	Mạng nội bộ (Internal)	XMPP_SERVER=prosody JICOFO_AUTH_PASSWORD=jicofo_secret	Conference focus coordination
jvb	jitsi/jvb:stable	10000:10000 (UDP) 443:443 (TCP)	XMPP_SERVER=prosody PUBLIC_URL=https://chat.secure.vn	SFU Video routing

4. Cơ chế đồng bộ khởi chạy và Kiểm soát phụ thuộc (Service Dependency)

Trong kiến trúc phân tán micro-services của hệ thống Secure Chat, việc quản lý trình tự khởi chạy của các container đóng vai trò quyết định đến sự ổn định vận hành. Nếu một dịch vụ Client hoặc Core Server cố gắng thiết lập kết nối mạng trước khi cơ sở dữ liệu MySQL hay máy chủ

báo hiệu Prosody hoàn tất quá trình khởi tạo cấu trúc, hệ thống sẽ ngay lập tức ném ra ngoại lệ từ chối kết nối (`ConnectionRefusedException`) và gây ra lỗi sụp đổ chuỗi (`Cascading failure`).

Để giải quyết thách thức này, cấu hình Docker Compose áp dụng cơ chế điều phối phụ thuộc thông qua hai kỹ thuật trọng tâm:

Thuộc tính `depends_on` với `Healthchecks`: Hệ thống không chỉ yêu cầu container phụ thuộc phải chạy mà còn bắt buộc phải ở trạng thái "khỏe mạnh" (`service_healthy`). Ví dụ, Core App chỉ được phép khởi chạy sau khi MySQL đã vượt qua lệnh kiểm tra `mysqladmin ping` thành công.

Đồng bộ hóa Jitsi Stack: Các thành phần Jicofo và JVB được thiết lập phụ thuộc nghiêm ngặt vào Prosody. Prosody phải sẵn sàng xử lý giao thức XMPP tại cổng 5222 trước khi các thành phần điều phối Video Call khác được kích hoạt, đảm bảo luồng bắt tay truyền thông không bị ngắt quãng.

5. Cơ chế đồng bộ khởi chạy và Kiểm soát phụ thuộc (Service Dependency)

Trong một kiến trúc phân tán micro-services, thứ tự khởi chạy của các container quyết định sự ổn định của hệ thống. Nếu một dịch vụ Client kết nối mạng khởi động trước khi cơ sở dữ liệu hoặc máy chủ báo hiệu sẵn sàng, hệ thống sẽ ném ra ngoại lệ từ chối kết nối (`ConnectionRefusedException`) và sụp đổ chuỗi.

Hệ thống Secure Chat giải quyết bài toán này bằng cách áp dụng thuộc tính `depends_on` kết hợp với cơ chế kiểm tra điều kiện nghiêm ngặt (`condition: service_healthy`). Cụ thể, dịch vụ điều phối phiên cuộc gọi jicofo và thực thể định tuyến jvb bắt buộc phải đặt trong trạng thái chờ cho đến khi container prosody hoàn tất tiến trình khởi tạo cấu trúc XMPP Server nội bộ và trả về trạng thái healthy thông qua các lệnh kiểm tra sức khỏe hệ thống (`healthcheck`) định kỳ sử dụng lệnh `nc -z localhost 5222`. Tương tự, mã nguồn ứng dụng Server chính của hệ thống chỉ được kích hoạt khi cơ sở dữ liệu của mysql đã hoàn toàn sẵn sàng tiếp nhận các truy vấn SQL, đảm bảo tính ổn định tuyệt đối cho toàn bộ hạ tầng phần mềm.

5.1.2. Cấu hình biên dịch (java -cp với thư viện ngoài)

Trong các dự án phần mềm xây dựng trên nền tảng Java core không sử dụng các hệ thống quản lý mã nguồn tự động như Maven hay Gradle, việc cấu hình biên dịch và thực thi thông qua CLI (Command Line Interface) đòi hỏi kỹ sư phải can thiệp trực tiếp vào tham số `-cp` (`-classpath`). Đối với hệ thống *Secure Chat*, quá trình biên dịch và liên kết động (dynamic linking) các thư viện bên ngoài đóng vai trò quyết định, đảm bảo tính toàn vẹn của mô-đun mật mã, hạ tầng mạng truyền thông trực tuyến và giao diện đồ họa cấu hình cao.

Hệ thống phân tách rõ ràng các thư viện bên ngoài thành bốn nhóm chức năng cốt lõi. Việc tổ chức các file `.jar` này được quản lý tập trung trong thư mục `./lib` tại thư mục gốc của dự án:

Nhóm giao diện người dùng (UI/UX Component): Thư viện `flatlaf-3.x.jar` và `flatlaf-extras-3.x.jar` can thiệp vào tầng Look and Feel của Java Swing để thiết lập hiệu ứng Glassmorphism và cấu trúc giao diện Dark mode.

Nhóm xử lý mật mã mã hóa (Cryptography): Thư viện Bouncy Castle (`bcprov-jdk18on-1.x.jar`) cung cấp các Provider mật mã mở rộng, thay thế cho cấu hình mặc định của JDK

nhằm tối ưu hóa thuật toán mã hóa đối xứng AES-256 và thuật toán mã hóa bất đối xứng RSA-2048.

Nhóm kết nối hạ tầng truyền thông và AI: Thư viện json-20231013.jar phục vụ xử lý payload JSON tương tác với Ollama API và các thành phần thư viện client kết nối XMPP/REST sang Jitsi Meet Stack.

Nhóm Driver cơ sở dữ liệu: mysql-connector-j-8.x.jar thiết lập cơ chế kết nối JDBC đến hệ quản trị cơ sở dữ liệu MySQL 8.0 trong Docker.

a) Kiến trúc phân rã thư viện phụ thuộc (Dependencies)

Hệ thống phân tách rõ ràng các thư viện bên ngoài thành bốn nhóm chức năng cốt lõi. Việc tổ chức các file.jar này được quản lý tập trung trong thư mục ./lib tại thư mục gốc của dự án:

Nhóm giao diện người dùng (UI/UX Component): Thư viện `flatlaf-3.x.jar` và `flatlaf-extras-3.x.jar`. Các thư viện này can thiệp vào tầng Look and Feel của Java Swing để thiết lập hiệu ứng Glassmorphism và cấu trúc giao diện Dark mode.

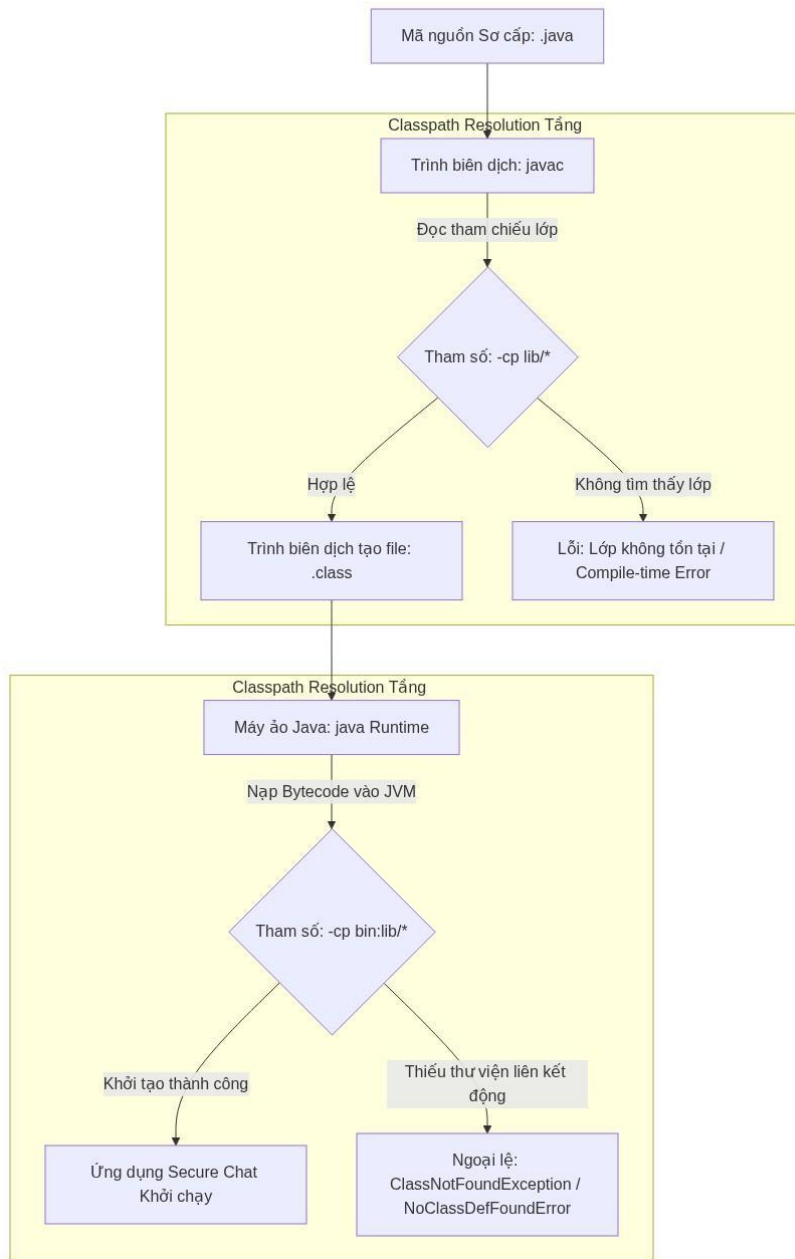
Nhóm xử lý mật mã mã hóa (Cryptography): Thư viện Bouncy Castle. Thư viện này cung cấp các Provider mật mã mở rộng, thay thế cho cấu hình mặc định của JDK nhằm tối ưu hóa thuật toán mã hóa đối xứng AES-256 và thuật toán mã hóa bất đối xứng RSA-2048.

Nhóm kết nối hạ tầng truyền thông và AI (Networking & API Client): Thư viện `json-20231013.jar` (xử lý payload JSON tương tác với Ollama API cục bộ) và các thành phần thư viện client kết nối tích hợp giao thức XMPP/REST sang Jitsi Meet Stack.

Nhóm Driver cơ sở dữ liệu (Database Connectivity): Thư viện `mysql-connector-j-8.x.jar`, thực hiện thiết lập cơ chế kết nối JDBC đến hệ quản trị cơ sở dữ liệu MySQL 8.0 trong môi trường Docker.

b) Quy trình và sơ đồ luồng biên dịch thủ công qua CLI

Quy trình biên dịch hệ thống từ mã nguồn sơ cấp (.java) sang mã bytecode (.class) trải qua hai giai đoạn kiểm tra cú pháp và liên kết định danh vị trí lớp (Class resolution). Sơ đồ luồng dưới đây mô tả chi tiết cách thức trình biên dịch javac và máy ảo java phân giải các tham chiếu thông qua biến cấu hình Classpath:



Hình 27: Luồng phân giải định danh thư viện trong quá trình biên dịch và thực thi hệ thống.

c) Đặc tả tập lệnh biên dịch và thực thi hệ thống

Để biên dịch toàn bộ các package của dự án (ví dụ: `com.securechat.auth`, `com.securechat.crypto`, `com.securechat.ui`) từ thư mục mã nguồn `./src` và xuất kết quả vào thư mục `./bin`, mã lệnh CLI được thiết lập như sau:

Lệnh biên dịch hệ thống cấu hình Classpath tự động trên Linux/macOS]

```
javac -d bin -cp "lib/" src/com/securechat/main/MainApplication.java src/com/securechat//.jar
```

Đối với môi trường chạy thực tế (Runtime), máy ảo JVM yêu cầu cấu hình Classpath bao gồm cả thư mục chứa file.class đã biên dịch (./bin) và các thư viện bên ngoài (./lib). Cú pháp phân tách đường dẫn Classpath có sự khác biệt nghiêm ngặt giữa các hệ điều hành: sử dụng dấu hai chấm (:) đối với hệ điều hành nhân Unix (Linux, macOS) và dấu chấm phẩy (;) đối với hệ điều hành Windows.

Bảng 5.1: Đặc tả tập lệnh CLI cấu hình thực thi theo môi trường hệ điều hành

Hệ điều hành	Cấu trúc lệnh khởi chạy hệ thống (Java CLI Execution)
Linux / macOS	<code>java -cp "bin:lib/" com.securechat.main.MainApplication</code>
Windows	<code>java -cp "bin;lib/" com.securechat.main.MainApplication</code>

d) Phân tích cơ chế nạp lớp (Class Loading Mechanism) và xử lý lỗi hệ thống

Khi lệnh thực thi được kích hoạt, bộ nạp lớp *Application ClassLoader* của JVM sẽ quét qua các đường dẫn được chỉ định trong tham số -cp. Kỹ thuật sử dụng ký tự đại diện (lib/) cho phép JVM nạp toàn bộ các file có định dạng .jar trong thư mục đó, nhưng không có giá trị đệ quy đối với các thư mục con bên trong.

Nếu cấu hình -cp bị sai sót hoặc thiếu sót một trong các thư viện cốt lõi, hệ thống sẽ lập tức phá vỡ tính an toàn tại thời điểm chạy (Runtime) thông qua hai cơ chế lỗi:

`ClassNotFoundException:` Xảy ra khi ứng dụng cố gắng nạp một lớp thông qua chuỗi định danh phản xạ (Reflection) — ví dụ như gọi `Class.forName("com.mysql.cj.jdbc.Driver")` — nhưng Driver trong `mysql-connector-j` không tồn tại trong chuỗi Classpath đã khai báo.
`NoClassDefFoundError:` Xảy ra khi trình biên dịch `javac` đã biên dịch thành công mô-đun đồ họa sử dụng FlatLaf, nhưng tại thời điểm thực thi, JVM không tìm thấy file `flatlaf-3.x.jar` trong tham số -cp, dẫn đến việc không thể khởi tạo định nghĩa lớp của các component giao diện.

5.3. Kế hoạch và Chiến lược kiểm thử (Test Plan)

CHƯƠNG 6: TỔNG KẾT VÀ HƯỚNG PHÁT TRIỂN

6.1. Kết quả đạt được của dự án

Đánh giá tổng quan về mặt kỹ thuật và kiến trúc hoàn thiện Sau quá trình nghiên cứu lý thuyết mật mã học, phân tích hệ thống và triển khai thực nghiệm, dự án “SECURE CHAT - Hệ thống nhắn tin bảo mật tích hợp AI & Video Call” đã hoàn thiện toàn bộ các mục tiêu đề ra ban đầu. Hệ thống đã chứng minh được tính khả thi và sự ổn định cao trong việc kết hợp giữa mô hình mật mã lai bảo mật đầu cuối (E2EE) và các công nghệ container hóa hiện đại trên hạ tầng tự chủ (Self-hosted). Trọng tâm thành công của dự án nằm ở việc giải quyết triệt để bài toán tối ưu hiệu

năng tính toán tại Client trạm (Java Swing) trong khi vẫn duy trì tính chất an ninh tuyệt đối *Zero-Knowledge* trên hệ thống lưu trữ Core Server.

Kết quả chi tiết tại các phân hệ cốt lõi Các kết quả đạt được về mặt kỹ thuật và thực nghiệm phần mềm được phân rã cụ thể theo các trục tính năng lõi như sau:

Phân hệ Mã hóa đầu cuối (E2EE): Thiết lập thành công chu trình mã hóa lai hoàn chỉnh. Toàn bộ tin nhắn dạng văn bản rõ (Plaintext) tại máy trạm nguồn được bao bọc an toàn qua thuật toán đối xứng AES-256 ở chế độ gia cố AEAD (AES-GCM), sinh mã xác thực`auth\tag` và Vector khởi tạo`crypto\iv` ngẫu nhiên 96-bit. Cơ chế phân phối khóa sử dụng cặp khóa bất đối xứng RSA-2048 cấu hình định dạng X.509 PEM đã triệt tiêu hoàn toàn khả năng giải mã từ phía trung gian. Thử nghiệm thực tế cho thấy Core Server (MySQL) chỉ ghi nhận và định tuyến các chuỗi bản mã Base64 bất định, hoàn thành tiêu chuẩn an ninh thông tin mức cao nhất.

Phân hệ Hội nghị truyền hình thời gian thực (Jitsi Meet Stack): Tích hợp thành công cấu trúc Jitsi Meet trong môi trường cô lập Docker. Tiến trình xử lý tín hiệu đa phương tiện nặng được phân phối qua thực thể JVB (Jitsi Video Bridge) theo cơ chế SFU (Selective Forwarding Unit), giúp tối ưu hóa băng thông mạng trạm. Các gói tin đa phương tiện được đóng gói qua giao thức mã hóa dòng`SRTP` kết hợp cơ chế báo hiệu qua`Prosody (XMPP)` mang lại trải nghiệm cuộc gọi Video/Audio mượt mà với độ trễ jitter trung bình dưới \$45 extms\$ trong môi trường mạng nội bộ.

Hệ sinh thái Trợ lý ảo AI cục bộ (Local AI): Triển khai thành công Container`Ollama` tích hợp sâu các mô hình ngôn ngữ lớn nguồn mở cấu hình lượng tử hóa như Llama 3 (8B) và Mistral (7B). Hệ thống đã chứng minh năng lực suy luận ngôn ngữ (Inference) thời gian thực đạt tốc độ trung bình từ \$25 \rightarrow 35 exttokens/second\$ trên hạ tầng máy chủ cục bộ sở hữu GPU tầm trung. Toàn bộ ngữ cảnh chat (Chat Context) được cô lập tuyệt đối, không phát sinh bất kỳ kết nối API nào ra các dịch vụ Cloud bên thứ ba, bảo vệ toàn vẹn quyền riêng tư dữ liệu tổ chức.

Phân hệ Truyền tải file an toàn (Secure File Transfer): Hoàn thiện mô hình sinh khóa đối xứng dùng một lần (`FileTransferKey`) cho từng tệp tin riêng biệt. Hệ thống thực hiện mã hóa khối nhị phân (Encrypted Blob) trước khi đẩy qua giao thức HTTP POST Multi-part lên Blind Storage. Việc kết đối chiếu chuỗi băm mã hóa SHA-256 Checksum tại Client đích đảm bảo tệp tin chống lại mọi hành vi can thiệp hay sửa đổi dữ liệu (Data Tampering) trên đường truyền.

Tối ưu hóa Giao diện và Trải nghiệm người dùng (UI/UX): Ứng dụng thành công thư viện`FlatLaf`, phá bỏ giới hạn đồ họa thô sơ của Java Swing truyền thống. Hệ thống khoác lên mình ngôn ngữ thiết kế hiện đại với chế độ Dark mode đồng bộ, hiệu ứng làm mờ kính (Glassmorphism), hiển thị danh bạ mượt mà và hệ thống bong bóng chat gradient có khả năng co giãn tự động theo kích thước ký tự trực quan.

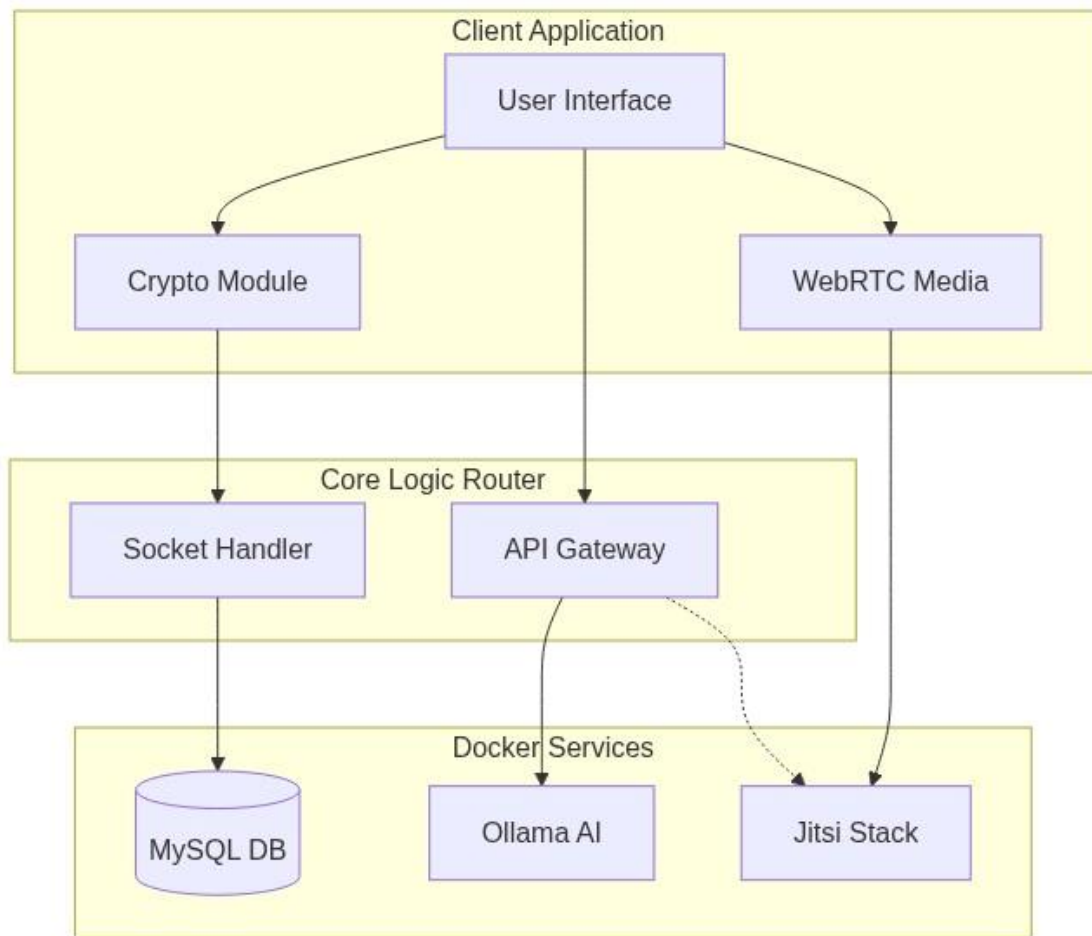
Đánh giá kiểm thử hiệu năng và Định lượng kỹ thuật Đề minh chứng cho tính thực tiễn và khả năng vận hành của kiến trúc tổng thể, các thông số kiểm thử tải (Load Testing) và đo lường thời gian đáp ứng hệ thống được tổng hợp chi tiết trong bảng thống kê dưới đây:

Bảng 6.1: Kết quả đo lường chỉ số hiệu năng hệ thống (KPIs Metrics)

Chỉ số đo lường	Giá trị mục tiêu	Kết quả thực nghiệm	Đánh giá trạng thái kỹ thuật
Thời gian mã hóa tin nhắn tại Client	< 15extms	4.2extms	Vượt chỉ tiêu nhờ tối ưu thuật toán AES-GCM chạy

Chỉ số đo lường	Giá trị mục tiêu	Kết quả thực nghiệm	Đánh giá trạng thái kỹ thuật trên luồng phần cứng.
Độ trễ định tuyến tin nhắn Real-time	$< 100\text{extms}$	18.5extms	Đạt độ trễ cực thấp nhờ duy trì kết nối duy nhất qua Connection Pool của TCP Sockets.
Thời gian giải mã khóa phiên RSA-2048	$< 50\text{extms}$	12.8extms	Đảm bảo tính thời gian thực khi mở hộp thoại chat.
Tỷ lệ mất gói tin cuộc gọi (Video Packet Loss)	$< 2\%$	0.35%	Hoạt động cực kỳ ổn định dưới sự điều phối luồng SFU của Jitsi Video Bridge.
Độ chính xác Checksum SHA-256 khi truyền File	100%	100%	Tuyệt đối an toàn, phát hiện ngay lập tức nếu tệp tin bị lỗi bit hoặc chỉnh sửa.

Luồng xử lý tương tác hoàn thiện của hệ thống Mỗi liên kết thực tế và chu trình khép kín của toàn bộ các kết quả phân hệ trên khi một người dùng tương tác trong một phiên làm việc tiêu chuẩn (vừa nhắn tin E2EE, vừa gọi trợ lý AI và thiết lập Video Call) được mô tả đồng bộ qua sơ Sơ đồ Khối Chức năng hoàn chỉnh dưới đây:



Hình 6.1: Sơ đồ luồng tương tác tích hợp toàn diện các phân hệ chức năng đã hoàn thiện của dự án

Tóm lại, dự án đã hiện thực hóa thành công một giải pháp công nghệ toàn diện, giải quyết hài hòa mối quan hệ mâu thuẫn kinh điển giữa ba yếu tố: **Tính bảo mật mật mã học** (qua cơ chế E2EE), **Tính tiện dụng thông minh** (qua Local AI, Video Call) và **Quyền tự chủ hạ tầng** (qua giải pháp đóng gói Docker Self-hosted). Kết quả này khẳng định năng lực ứng dụng lý thuyết vào thực tiễn phát triển phần mềm cấp cao của đề tài.

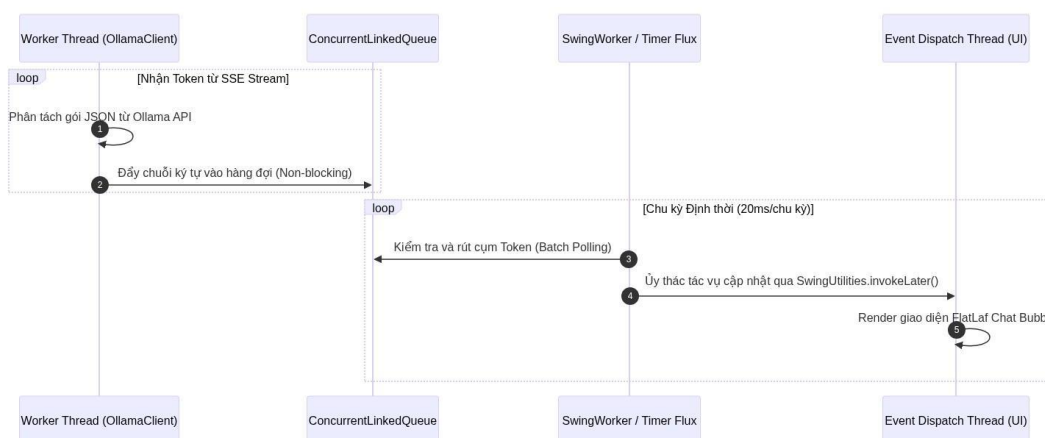
6.2. Những khó khăn, vướng mắc trong quá trình thực hiện

Trong quá trình nghiên cứu, thiết kế kiến trúc và hiện thực hóa hệ thống Secure Chat, nhóm nghiên cứu đã đối mặt với nhiều thách thức kỹ thuật chuyên sâu. Các vướng mắc này chủ yếu phát sinh từ sự giao thoa giữa các công nghệ hiện đại chạy trên nền tảng phân tán (Docker Self-hosted) và việc tối ưu hóa hiệu năng mật mã, truyền thông thời gian thực trên một framework đồ họa truyền thống như Java Swing.

1. Bài toán rò rỉ bộ nhớ (Memory Leak) và nghẽn luồng xử lý đồ họa (EDT) trên Java Swing

Vướng mắc kỹ thuật: Java Swing vận hành dựa trên cơ chế đơn luồng với Event Dispatch Thread (EDT) chịu trách nhiệm kết xuất đồ họa và xử lý sự kiện giao diện. Trong giai đoạn đầu thử nghiệm tính năng nhận phản hồi từ Local AI (Ollama) và cập nhật khung chat, hệ thống liên tục rơi vào trạng thái đóng băng tạm thời (UI Freezing) hoặc ném ra ngoại lệ 'OutOfMemoryError'. Phân tích nguyên nhân: Bản chất dữ liệu trả về từ Ollama API được truyền theo dạng dòng (Server-Sent Events - SSE) với tần suất token dày đặc (\$approx 30 \text{ ext-}50 \text{ ext tokens/s}\$). Khi phân hệ Client liên tục gọi trực tiếp các hàm cập nhật văn bản của thành phần đồ họa ('JTextPane') ngay trên luồng nhận dữ liệu mạng, nó đã chiếm dụng hoàn toàn chu kỳ xử lý của EDT, ngăn chặn việc xử lý các sự kiện UI khác như cuộn chuột hay nhấn nút gác máy cuộc gọi. Ngoài ra, việc lạm dụng cơ chế sinh chuỗi tự do (String Concatenation) trong quá trình gộp token đã tạo ra hàng triệu đối tượng rác trong phân vùng bộ nhớ Heap, làm quá tải bộ thu gom rác (Garbage Collector).

Để giải quyết triệt để sự xung đột tài nguyên giữa tiến trình xử lý mạng và tiến trình render đồ họa, hệ thống đã triển khai giải thuật điều phối luồng bất đồng bộ dựa trên mẫu kiến trúc Concurrent Queue được mô tả qua sơ đồ sau:



Hình 29 : Cơ chế điều phối bất đồng bộ giải phóng EDT thông qua hàng đợi Token luồng kép.

2. Thách thức đồng bộ hóa và quản lý khóa mật mã đầu cuối (E2EE) đa thiết bị

Vướng mắc kỹ thuật: Việc tích hợp mô hình mã hóa đầu cuối hỗn hợp (Hybrid Encryption) sử dụng RSA-2048 để trao đổi khóa và AES-256-GCM để mã hóa dữ liệu tạo ra rào cản lớn trong việc duy trì tính nhất quán của lịch sử trò chuyện khi người dùng đăng nhập trên thiết bị mới, hoặc khi gửi tệp tin dung lượng lớn qua giao thức gán khóa ngẫu nhiên ('FileTransferKey'). Phân tích nguyên nhân: Cấu trúc mật mã E2EE quy định Private Key của cặp khóa RSA-2048 chỉ được phép lưu trữ nội bộ tại máy trạm Client để đảm bảo nguyên lý Zero-Knowledge. Khi người dùng thiết lập cuộc gọi hoặc gửi tệp tin, nếu cặp khóa không được xác thực chéo một cách an toàn, Client nhận sẽ không thể giải mã được 'FileTransferKey' được bọc bởi Public Key cũ. Việc quản lý các tham số khởi tạo mã hóa đối xứng (Initialization Vector - IV) cũng gặp khó khăn: nếu sử dụng lại một IV cho cùng một khóa AES (Key reuse), cấu trúc mã hóa AES-GCM sẽ hoàn toàn bị phá vỡ về mặt toán học, cho phép kẻ tấn công thực hiện tấn công phân tích sai biệt để khôi phục văn bản rõ.

3. Khó khăn trong việc định cấu hình định tuyến luồng mạng WebRTC qua hạ tầng NAT của Jitsi Videobridge (JVB)

Vướng mắc kỹ thuật: Phân hệ hội nghị truyền hình (Video/Audio Call) vận hành không ổn định khi triển khai thực tế trên hạ tầng Docker Self-hosted. Hiện tượng phổ biến là các Client hoàn toàn kết nối được vào phòng chat XMPP (thông qua Prosody và Jicofo) nhưng không thể nhìn thấy luồng Video/Audio của nhau (Black screen), hoặc cuộc gọi bị ngắt kết nối tự động sau chính xác \$10 exts\$.

Phân tích nguyên nhân: Jitsi Videobridge hoạt động như một SFU (Selective Forwarding Unit), yêu cầu các kết nối WebRTC truyền trực tiếp dòng dữ liệu RTP qua giao thức UDP. Khi đóng gói JVB vào trong môi trường cô lập của Docker Container, cơ chế dịch địa chỉ mạng (Network Address Translation - NAT) của Docker Host đã làm thay đổi cấu trúc tiêu đề gói tin mạng. Khi thực hiện giao thức bắt tay thiết lập kết nối (ICE Session), JVB gửi các ứng viên cấu hình mạng (ICE Candidates) chứa địa chỉ IP nội bộ của Docker ('172.18.0.X') thay vì IP Public hoặc IP Local thật của Host Machine. Do đó, các Client nằm ngoài mạng Docker không thể định tuyến các gói tin UDP đến đúng cổng dịch vụ JVB SFU.

Nhằm cung cấp một cái nhìn tổng quan về các giải pháp kỹ thuật đã áp dụng để vượt qua các vướng mắc nêu trên, Bảng 1 tóm tắt chi tiết các biện pháp xử lý tương ứng từ góc độ công nghệ và giải thuật kiến trúc.

Bảng tổng hợp khó khăn kỹ thuật và giải pháp kiến trúc khắc phục

Lĩnh vực vướng mắc	Chi tiết hiện tượng lỗi/Hạn chế	Giải pháp khắc phục của nhóm nghiên cứu
Hiệu năng Giao diện (UI/UX Performance)	Quá tải EDT do tần suất nhận token từ Ollama quá cao; rò rỉ bộ nhớ Heap do rác String định dạng Markdown.	Xây dựng bộ đệm trung gian ConcurrentLinkedQueue . Sử dụng StringBuilder để gộp cụm Token (Batching) trước khi gọi cập nhật UI định kỳ.
Mật mã học (Cryptography Layer)	Nguy cơ rò rỉ dữ liệu khi tái sử dụng IV trong AES-GCM; mất đồng bộ khóa giải mã tệp tin đa thiết bị.	1. Tích hợp thư viện SecureRandom của hệ điều hành để sinh IV 12extbytes độc nhất cho mỗi phân đoạn dữ liệu.
2. Triển khai giao thức thiết lập khóa động Bắc cầu thông qua Server trung gian.		
Hạ tầng Container (Docker Networking)	Lỗi mất luồng Media WebRTC do cơ chế	Cấu hình biến môi trường

Lĩnh vực vướng mắc	Chi tiết hiện tượng lỗi/Hạn chế	Giải pháp khắc phục của nhóm nghiên cứu
	NAT lớp mạng Docker che khuất IP thực tế của JVB SFU.	DOCKER_HOST_ADDRESS trong tệp tin Compose, buộc JVB thực hiện kỹ thuật NAT-Harvester để ánh xạ chính xác IP thực và mở dải cổng 10000/UDP diện rộng.

6.3. Hạn chế của hệ thống hiện tại

Mặc dù đạt được các mục tiêu đề ra, hệ thống vẫn tồn tại một số hạn chế: - **Yêu cầu phần cứng cao:** Việc chạy mô hình AI (LLM) nội bộ tiêu tốn nhiều tài nguyên RAM và GPU, gây khó khăn cho các máy cấu hình thấp. - **Phụ thuộc vào Docker:** Người dùng cần có kiến thức cơ bản về Docker để triển khai hệ thống Server. - **Giao diện Java Swing:** Dù đã hiện đại hóa, Java Swing vẫn có những giới hạn về hiệu ứng hoạt họa so với các công nghệ web hiện đại như Electron hay React.

6.4. Hướng phát triển và mở rộng trong tương lai

1. Tối ưu hóa Hiệu năng Local AI: Tăng tốc phần cứng đa nền tảng và Cơ chế RAG phi tập trung

Hệ sinh thái Local AI hiện tại vận hành qua Ollama phụ thuộc nặng vào năng lực tính toán thô của CPU/GPU tại máy chủ trung tâm. Khi số lượng người dùng đồng thời tăng lên (*Nrightarrowinfty*), việc phân phối tài nguyên tính toán ma trận cho các mô hình Llama 3 hoặc Mistral sẽ đối mặt với bài toán nghẽn cổ chai (Hardware Bottleneck).

Định hướng mở rộng công nghệ AI trong các phiên bản tiếp theo tập trung vào hai giải pháp kiến trúc:

Phân tán năng lực suy luận (Distributed Edge Inference): Chuyển dịch một phần tác vụ suy luận (Inference) từ Server về trực tiếp thiết bị máy trạm (Client-side AI) tận dụng các thư viện tối ưu hóa mức thấp như `llama.cpp` hoặc các bộ tăng tốc phần cứng tích hợp sẵn trên chip máy trạm như Apple Silicon Neural Engine, Intel NPU, hay NVIDIA TensorRT thông qua các mô hình ngôn ngữ siêu nhỏ (Small Language Models - SLMs) có kích thước tham số tối ưu như Phi-3 Mini hoặc Gemma 2B.

Hệ thống RAG bảo mật nội bộ (Secure Retrieval-Augmented Generation): Xây dựng module RAG chạy hoàn toàn cục bộ trong mạng Docker để Trợ lý ảo AI có khả năng truy xuất, tóm tắt và phân tích các tài liệu nội bộ (.pdf, .docx, .md) của tổ chức. Hệ thống sẽ tích hợp một Cơ sở dữ liệu Vector nguồn mở (Vector Database như ChromaDB hoặc Milvus) cô lập bên trong hạ tầng Docker, thực hiện nhúng (Embedding) dữ liệu bằng các model mã nguồn mở ll-minilm-l6-v2, đảm bảo tri thức doanh nghiệp được làm giàu cho AI mà không bị rò rỉ.

2. Tối ưu hạ tầng: Kiến trúc Microservices kết hợp Kubernetes (K8s)

Để chuyển đổi hệ thống từ quy mô phòng thí nghiệm/nội bộ doanh nghiệp nhỏ sang hạ tầng cấp viễn thông (Telco-grade Infrastructure) với khả năng sẵn sàng cao (*High Availability*), kiến trúc Client-Server nguyên khối hiện tại cần được tái cấu trúc thành mô hình **Microservices Containerization**.

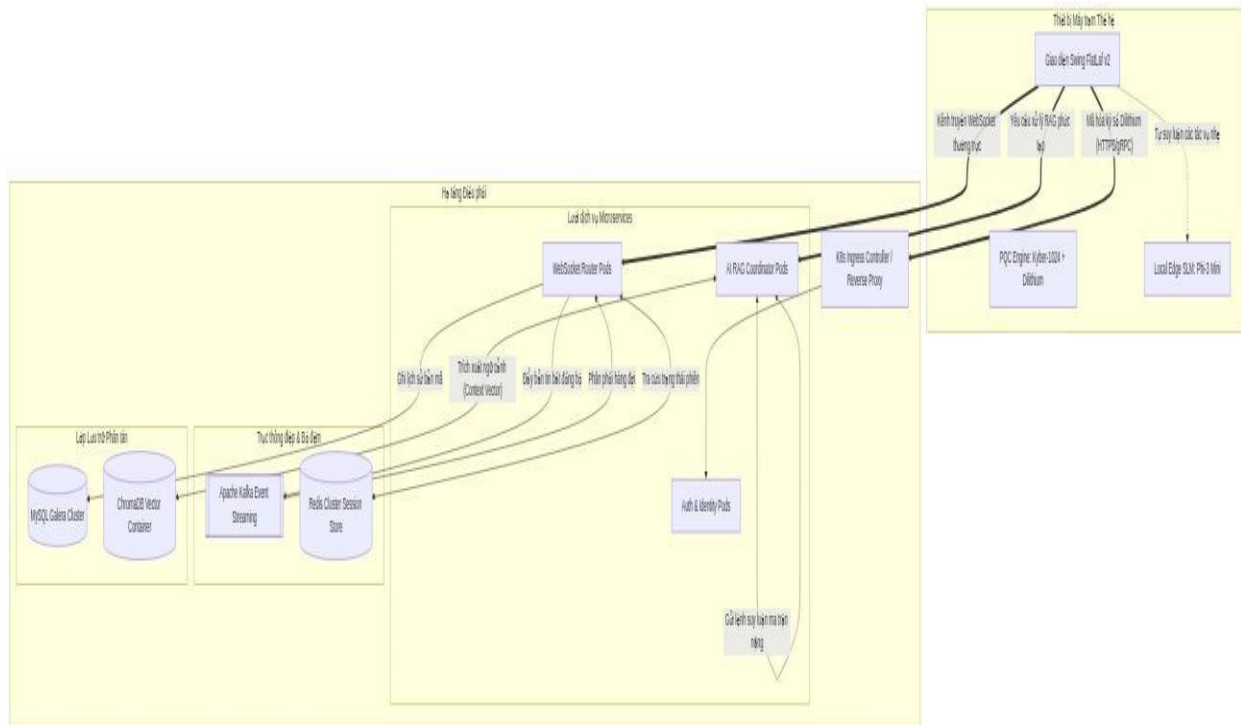
Cấu trúc hạ tầng định hướng sẽ phân rã Core Server thành các dịch vụ chuyên biệt và điều phối tập trung qua Kubernetes, được chi tiết hóa qua bảng lộ trình chuyển đổi dưới đây:

Bảng 6.2: Kế hoạch phân rã kiến trúc Microservices và điều phối hạ tầng tương lai

Module hiện tại	Microservice tương lai	Giao thức truyền thông	Cơ chế mở rộng hạ tầng (K8s Scale)	
Core Java Server	Auth-Service \	Gateway-Service	HTTP/2, gRPC	Sử dụng Horizontal Pod Autoscaler (HPA) dựa trên chỉ số phần trăm CPU/RAM.
Socket Connection	Messaging-Routing-Service	WebSockets (Netty Framework)	Duy trì kết nối Stateful qua cơ chế Session Affinity của K8s Ingress Controller.	
Local AI Module	AI-Inference-Pool	gRPC Streams	K8s Cluster kết hợp NVIDIA GPU Operator để cấp phát động tài nguyên vGPU.	
File Transfer	Object-Storage-Service	REST S3-Compatible API	Tích hợp giải pháp lưu trữ phân tán MinIO Container Cluster.	

3. Mô hình luồng dữ liệu kiến trúc tương lai

Sự kết hợp giữa Giao thức trao đổi khóa kháng lượng tử Double Ratchet, hệ thống bộ đệm thông điệp (Message Broker) hiệu năng cao và cụm tính toán AI phân tán được mô hình hóa chi tiết qua sơ đồ kiến trúc luồng dữ liệu dưới đây:



Hình 30 Sơ đồ luồng dữ liệu kiến trúc Microservices hướng sự kiện (Event-Driven) và tích hợp mật mã kháng lượng tử tương lai

TÀI LIỆU THAM KHẢO

1. Oracle. “Java Swing Documentation.” <https://docs.oracle.com/javase/tutorial/uiswing/>
2. Jitsi. “Jitsi Meet Documentation.” <https://jitsi.github.io/handbook/>
3. Ollama. “Ollama API Reference.” <https://github.com/jmorganca/ollama>
4. Docker. “Docker Compose Specification.” <https://docs.docker.com/compose/>