TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
TRƯỜNG ĐIỆN ĐIỆN TỬ

****

BÁO CÁO BÀI TẬP LỚN

Đề tài: Ứng dụng Chat Nhóm WPF sử dụng kiến trúc Model - View – ViewModel(MVVM)

Môn học: Lập trình nâng cao
Mã lớp: 157189 - ET4430E

Giảng viên	**PhD. Vũ Song Tùng**	
Sinh viên	**Vũ Minh Hiển**	**20224311**
	Bùi Hoàng Giang	**20224307**
	Đinh Quang Hiển	**20224281**
	Nguyễn Huy Hoàng	**20224313**

Hà Nội, tháng 06 năm 2025

MỤC LỤC

MỞ ĐẦU

Báo cáo này trình bày quá trình phát triển một ứng dụng **chat nhóm thời gian thực trên nền tảng desktop**, cho phép nhiều người dùng kết nối qua mạng nội bộ hoặc Internet để trò chuyện, chia sẻ tệp tin và tương tác trực tuyến. Ứng dụng được xây dựng bằng công nghệ **WPF** kết hợp với mô hình kiến trúc **MVVM (Model - View - ViewModel)** trên nền tảng **.NET**, cho phép tách biệt rõ ràng giữa giao diện người dùng và logic xử lý, đồng thời hỗ trợ cập nhật giao diện tự động khi dữ liệu thay đổi.

Trong bối cảnh làm việc và học tập từ xa ngày càng phổ biến, các phần mềm chat nhóm đóng vai trò quan trọng trong việc hỗ trợ giao tiếp nhanh chóng và hiệu quả. Mặc dù có nhiều ứng dụng phổ biến trên thị trường như Discord, Microsoft Teams hay Slack, việc xây dựng một hệ thống riêng giúp người học hiểu rõ nguyên lý hoạt động của các ứng dụng thời gian thực, từ giao tiếp mạng, xử lý bất đồng bộ, đến cập nhật giao diện theo mô hình hiện đại.

Dự án tập trung vào việc xây dựng một hệ thống **Client-Server** sử dụng **TCP Socket** để truyền dữ liệu, hỗ trợ nhiều tính năng thực tế như: gửi và nhận tin nhắn, chia sẻ tệp tin (hình ảnh, tài liệu), trả lời tin nhắn cụ thể (reply), hiển thị danh sách người dùng đang hoạt động và phản hồi giao diện theo thời gian thực. Đồng thời, kiến trúc MVVM giúp ứng dụng dễ mở rộng, kiểm thử và bảo trì.

Báo cáo này sẽ trình bày chi tiết về mục tiêu, công nghệ sử dụng, thiết kế hệ thống, cách triển khai logic xử lý, kết quả đạt được cũng như những hướng phát triển tiếp theo trong tương lai.

1. Mục đích đề tài

Mục tiêu chính của đề tài là xây dựng một **ứng dụng trò chuyện nhóm thời gian thực** trên nền tảng **desktop**, sử dụng công nghệ **WPF** và **.NET**, kết hợp mô hình kiến trúc **MVVM (Model - View - ViewModel)**. Đề tài hướng đến việc:

- **Áp dụng kiến thức về lập trình socket** trong môi trường .NET để hiện thực hóa kênh truyền thông giữa các client và server.
- **Thiết kế và xây dựng giao diện người dùng hiện đại, trực quan bằng WPF**, có khả năng phản hồi tốt và hỗ trợ cập nhật theo thời gian thực.
- **Tổ chức mã nguồn theo mô hình MVVM**, nhằm phân tách rõ ràng giữa logic nghiệp vụ, logic giao diện và phần hiển thị.
- Tích hợp các tính năng mở rộng như: gửi tệp tin (ảnh, tài liệu), trả lời tin nhắn, hiển thị danh sách người dùng đang hoạt động và nhắc tên người dùng.

Thông qua quá trình xây dựng, nhóm sinh viên có cơ hội rèn luyện kỹ năng lập trình bất đồng bộ, xử lý đa luồng, phân tích hệ thống, cũng như tổ chức kiến trúc phần mềm theo mô hình chuẩn.

2. Nhiệm vụ của đề tài

2.1. Thiết kế giao diện người dùng

Thiết kế giao diện trò chuyện theo bố cục hợp lý, bao gồm:

- Vùng nhập và hiển thị tin nhắn.
- Danh sách người dùng đang hoạt động.
- Giao diện gửi và xem trước tệp tin.
- Vùng hiển thị tin nhắn dạng reply hoặc hệ thống.

Giao diện được xây dựng bằng WPF, kết hợp với **DataBinding** và **ObservableCollection** để phản hồi dữ liệu theo thời gian thực, đảm bảo cập nhật động mà không cần thao tác thủ công.

2.2. Thiết kế giao thức truyền thông

Thiết kế **giao thức truyền tin tùy chỉnh** giữa client và server, sử dụng **TCP socket**. Các loại bản tin được phân biệt qua tiền tố như _user_, _file_, _reply_, giúp cả hai bên hiểu được cấu trúc dữ liệu cần xử lý. Việc mã hóa tệp tin được thực hiện bằng **Base64**, đảm bảo tính toàn vẹn khi truyền dữ liệu phi văn bản qua luồng socket.

2.3. Phát triển các cơ chế của 1 ứng dụng chat

Phát triển các logic xử lý sau:

- Quản lý kết nối: Server lắng nghe client, lưu trữ danh sách người dùng, phát sóng tin nhắn đến toàn bộ client.
- Gửi/nhận tin nhắn: Client gửi dữ liệu văn bản, ảnh, hoặc lệnh trả lời tới server, server phản hồi lại toàn bộ client.
- Giao tiếp bất đồng bộ: Toàn bộ hệ thống vận hành không đồng bộ, đảm bảo không nghẽn luồng chính kể cả khi có nhiều kết nối đồng thời.
- Cập nhật giao diện an toàn từ luồng nền bằng Dispatcher.Invoke().

2.4. Phân công nhiệm vụ thành viên

- Vũ Minh Hiển: Thiết kế giao diện người dùng, tổ chức cấu trúc View + ViewModel theo mô hình MVVM.
- Bùi Hoàng Giang: Xây dựng logic xử lý và truyền thông ở phía Server.
- Đinh Quang Hiển & Nguyễn Huy Hoàng: Phát triển toàn bộ luồng xử lý phía Client, bao gồm nhận dữ liệu, xử lý tương tác và liên kết dữ liệu với ViewModel.

CHƯƠNG 1: TỔNG QUAN VỀ HỆ THỐNG CHAT NHÓM WPF

1.1. Giới thiệu tổng quan về hệ thống chat nhóm

Hệ thống chat nhóm là một trong những dạng ứng dụng phổ biến và thiết yếu trong các hoạt động giao tiếp hiện đại, đặc biệt trong môi trường học tập, làm việc nhóm hoặc tổ chức từ xa. Mục tiêu của ứng dụng là tạo ra một nền tảng **trò chuyện thời gian thực**, hỗ trợ **giao tiếp nhiều người dùng đồng thời**, thông qua các chức năng cơ bản như gửi tin nhắn, chia sẻ tệp tin, trả lời tin nhắn, cập nhật danh sách người dùng đang hoạt động và phản hồi giao diện tức thời.

Ứng dụng được phát triển trên nền tảng **.NET Framework** sử dụng **WPF (Windows Presentation Foundation)** để xây dựng giao diện desktop hiện đại và thân thiện. Bên cạnh đó, hệ thống được tổ chức theo mô hình kiến trúc **MVVM (Model - View - ViewModel)** giúp phân tách rõ ràng giữa giao diện, dữ liệu và logic xử lý, từ đó nâng cao khả năng bảo trì, mở rộng và kiểm thử của hệ thống.

Toàn bộ hệ thống vận hành dựa trên mô hình **Client-Server** sử dụng **TCP Socket**, trong đó server đóng vai trò trung tâm điều phối và truyền tin, còn client là nơi người dùng trực tiếp tương tác. Giao thức truyền thông được thiết kế riêng, đơn giản và dễ mở rộng, sử dụng tiền tố định danh và ký tự phân tách để truyền các loại nội dung khác nhau như văn bản, tệp tin hoặc phản hồi.

1.2. Các công nghệ sử dụng

1.2.1. WPF (Windows Presentation Foundation)

WPF là nền tảng giao diện người dùng mạnh mẽ được tích hợp trong .NET Framework, hỗ trợ thiết kế UI với khả năng tùy biến cao. WPF hỗ trợ các tính năng như **Data Binding**, **Commanding**, **Styles/Templates**, và đặc biệt là **MVVM**, giúp xây dựng các ứng dụng giàu tính tương tác với cấu trúc rõ ràng.

Trong ứng dụng chat nhóm, WPF được sử dụng để:

- Xây dựng các thành phần giao diện như khung chat, khung danh sách người dùng, khung xem trước tệp tin.
- Hiển thị tin nhắn với định dạng đa dạng (text, reply, system, file) bằng `DataTemplate`.
- Tự động cập nhật giao diện khi dữ liệu thay đổi nhờ `ObservableCollection<T>` và `INotifyPropertyChanged`.

1.2.2. MVVM (Model - View - ViewModel)

MVVM là mô hình thiết kế giúp tách biệt phần giao diện người dùng (View), logic xử lý giao diện (ViewModel), và dữ liệu nghiệp vụ (Model). Trong ứng dụng này:

- **Model** đại diện cho dữ liệu và cấu trúc tin nhắn, người dùng, tệp tin.
- **ViewModel** xử lý toàn bộ logic: kết nối, gửi/nhận dữ liệu, phân tích cú pháp, cập nhật danh sách người dùng, và cung cấp dữ liệu cho View.
- **View** được định nghĩa trong XAML, binding trực tiếp với ViewModel để hiển thị dữ liệu.

Việc sử dụng MVVM giúp code dễ đọc, dễ kiểm thử và mở rộng khi cần thêm tính năng.

1.2.3. TCP Socket (System.Net.Sockets)

TCP là giao thức truyền tin **hướng kết nối** và **đáng tin cậy**, được sử dụng trong hệ thống để:

- Thiết lập kết nối giữa client và server.
- Truyền tải các bản tin bao gồm văn bản, tệp tin dưới dạng chuỗi.
- Duy trì trạng thái kết nối ổn định trong toàn bộ phiên làm việc.

Server sử dụng TcpListener để lắng nghe kết nối tại một cổng cụ thể (mặc định 8888), trong khi client sử dụng TcpClient để gửi/nhận dữ liệu dưới dạng dòng văn bản (StreamWriter và StreamReader).

1.2.4. Bất đồng bộ và luồng giao diện (Dispatcher)

Toàn bộ hệ thống xử lý mạng được triển khai dưới dạng **bất đồng bộ (async/await)**, giúp đảm bảo không gây nghẽn UI thread. Khi có dữ liệu đến từ server, client sử dụng Dispatcher.Invoke() để đảm bảo các thao tác cập nhật giao diện được thực hiện đúng luồng.

1.2.5. Mã hóa Base64

Để truyền tệp tin (đặc biệt là ảnh) qua TCP dưới dạng văn bản, hệ thống sử dụng cơ chế mã hóa và giải mã **Base64**. Khi người dùng gửi tệp, nội dung file được chuyển thành chuỗi Base64, gắn với thông tin định danh tệp (tên file, loại nội dung) và truyền đi qua mạng.

1.3. Kiến trúc hệ thống

Hệ thống được thiết kế theo mô hình **Client-Server cổ điển**, trong đó:

- **Client (WpfChatApp)**:
  - Là ứng dụng desktop được xây dựng bằng WPF.
  - Cho phép người dùng kết nối, gửi và nhận tin nhắn/tệp tin, trả lời tin nhắn, và tương tác thông qua giao diện MVVM.
  - Tự động cập nhật danh sách người dùng đang hoạt động.
- **Server (ChatServer)**:
  - Là một ứng dụng console đơn giản, sử dụng TcpListener để lắng nghe các kết nối mới.
  - Quản lý danh sách người dùng đang hoạt động bằng Dictionary<TcpClient, ClientInfo>.
  - Phát tán (broadcast) các bản tin đến toàn bộ client theo logic đồng thời (multi-client support).
- **Giao thức truyền thông**:
  - Dữ liệu truyền giữa hai bên dưới dạng chuỗi văn bản, phân tách bằng ký tự |.
  - Các loại bản tin gồm có: _user_ (tin nhắn thường), _file_ (tệp tin), _reply_ (phản hồi), _system_ (thông báo hệ thống), _userlist_ (danh sách người dùng).
  - Client và server đều sử dụng cùng một quy tắc phân tích chuỗi để trích xuất dữ liệu.

CHƯƠNG 2: PHÂN TÍCH THIẾT KẾ HỆ THỐNG

2.1. Phân tích yêu cầu

Hệ thống cần đáp ứng các yêu cầu chức năng và phi chức năng sau:

- Yêu cầu chức năng:

- **F1:** Hệ thống phải cho phép nhiều người dùng kết nối vào một phòng chat chung.
- **F2:** Người dùng phải có thể đặt một tên định danh (Nickname) trước khi tham gia.
- **F3:** Hệ thống phải hiển thị danh sách tất cả người dùng đang online trong phòng.
- **F4:** Người dùng phải có thể gửi và nhận tin nhắn văn bản trong thời gian thực.
- **F5:** Hệ thống phải phân biệt tin nhắn của những người dùng khác nhau bằng màu sắc riêng.
- **F6:** Người dùng phải có thể trả lời (reply) một tin nhắn cụ thể của người khác.
- **F7:** Người dùng phải có thể gửi các tệp tin (giới hạn dưới 2MB) cho cả phòng chat.

- **F8:** Người dùng phải có thể lưu một tệp tin đã nhận về máy tính của mình.
- **F9:** Người dùng phải có thể chèn các biểu tượng cảm xúc (emoji) đơn giản vào tin nhắn.
- **F10:** Hệ thống phải hiển thị các thông báo khi có người tham gia hoặc rời khỏi phòng.

- Yêu cầu phi chức năng:
  - **NF1:** Giao diện người dùng phải trực quan, hiện đại và dễ sử dụng.
  - **NF2:** Tin nhắn phải được gửi và nhận gần như ngay lập tức trong điều kiện mạng LAN ổn định.
  - **NF3:** Ứng dụng client phải chạy được trên hệ điều hành Windows.
  - **NF4:** Hệ thống phải xử lý được việc người dùng ngắt kết nối đột ngột mà không làm ảnh hưởng đến những người khác.
  - **NF5 (Kiến trúc):** Mã nguồn phía Client phải được cấu trúc theo mô hình MVVM để tách biệt giao diện (View) và logic (ViewModel), tăng khả năng bảo trì và kiểm thử.

2.2. Thiết kế cơ sở dữ liệu

Ứng dụng không sử dụng cơ sở dữ liệu vật lý. Thay vào đó, dữ liệu được quản lý **tạm thời trên bộ nhớ** cả phía Client và Server.

- **Phía Server**: sử dụng **Dictionary<TcpClient, ClientInfo>** để lưu danh sách người dùng kết nối.
- **Phía Client**:
  - **ObservableCollection<ChatMessage>** để quản lý danh sách tin nhắn.
  - **ObservableCollection<string>** để lưu danh sách người dùng đang online.
  - **ChatMessage** là lớp đại diện cho một tin nhắn, có thể là văn bản, tệp, hoặc reply.

Cấu trúc lớp **ChatMessage:**

SenderName: Tên người gửi tin nhắn. Dùng để hiển thị tên bên cạnh nội dung chat. Nếu là tin nhắn hệ thống thì là "System".

SenderColor: Màu sắc dùng để phân biệt người gửi. Thường gán ngẫu nhiên cho mỗi user khi kết nối.

Content: Nội dung văn bản của tin nhắn. Đây là phần chính được hiển thị trên giao diện.

IsSentByMe: Xác định xem tin nhắn này có phải do chính người dùng gửi không. Nếu đúng thì canh phải (Right), ngược lại là tin nhận (canh trái).

IsReply: Đánh dấu tin nhắn này là một lời phản hồi đến một tin nhắn khác.

RepliedToUsername: Tên người đã gửi tin nhắn gốc mà người dùng đang trả lời. Hiển thị phía trên nội dung chính.

RepliedToContent: Nội dung của tin nhắn gốc đang được phản hồi. Nếu tin gốc là tệp, giá trị này sẽ là "Tệp: <Tên tệp>".

IsFile: Xác định đây có phải là tin nhắn chứa tệp tin không. Nếu true, hệ thống sẽ xử lý và hiển thị như tệp/tệp ảnh.

FileName: Tên gốc của tệp được gửi, dùng để hiển thị hoặc khi lưu lại.

FileContentBase64: Dữ liệu của tệp, đã được mã hóa sang định dạng Base64 để gửi qua TCP.

IsImagePreview: Nếu true, nghĩa là tệp này là ảnh và có thể hiển thị preview trong UI (không cần tải về).

2.3. Thiết kế luồng hoạt động chính

2.3.1. Luồng tạo và tham gia phòng chat

1. **Người dùng nhập tên** và nhấn nút "Kết nối".
2. MainWindowViewModel gọi ConnectAsync(), tạo TcpClient, kết nối đến Server (dùng IP/Port).
3. Gửi tên người dùng đầu tiên đến Server (_writer.WriteLineAsync(username)).
4. **Server** nhận kết nối mới, đọc tên, thêm vào danh sách.

Server:

5. Gửi bản tin hệ thống _system_ tới toàn bộ client (ví dụ: "Hiển thị A đã tham gia").
6. Gửi _userlist_ (danh sách tên người dùng) tới toàn bộ client.
7. Client xử lý _userlist_, cập nhật danh sách hiển thị người dùng online.
8. Từ đó, client bắt đầu vòng ListenForMessagesAsync(), liên tục lắng nghe các bản tin từ server và hiển thị lên giao diện.

2.3.2. Luồng gửi tin nhắn thường

1. Người dùng nhập nội dung và nhấn nút gửi.
2. SendCommand gọi **SendMessageAsync()**.
3. Nếu đang trả lời tin nhắn, định dạng tin nhắn dạng **_reply_|...**, ngược lại là tin nhắn thường.
4. Gửi tin nhắn lên server qua **_writer.WriteLineAsync**.
5. Server nhận, đóng gói lại thành **_user_** hoặc **_reply_** và broadcast cho tất cả client.
6. Client nhận, hiển thị tin nhắn lên giao diện.

2.3.3. Luồng trả lời tin nhắn

1. Người dùng chọn tin nhắn để trả lời.
2. ReplyCommand gọi **ReplyToMessage()**, hiển thị khung trả lời.
3. Người dùng nhập nội dung và gửi.
4. **SendMessageAsync()** gửi tin nhắn dạng **_reply_|user|content|message**.
5. Server nhận, đóng gói lại thành **_reply_** và broadcast.
6. Client nhận, hiển thị tin nhắn dạng trả lời.

2.3.4. Luồng gửi tệp tin

1. Người dùng nhấn nút **Đính kèm file**.
2. AttachFileCommand gọi AttachFileAsync().
3. Hệ thống mở hộp thoại chọn file (OpenFileDialog).
4. Người dùng chọn file → chương trình kiểm tra kích thước.
5. Nếu hợp lệ, nội dung file được đọc và mã hóa base64.
6. Bản tin _file_|Tên|Base64 được gửi tới server.
7. Server nhận và broadcast đến tất cả client.
8. Client nhận bản tin:
    - Nếu là ảnh: hiển thị preview.
    - Nếu là file khác: hiển thị tên file và nút tải về.

2.3.5. Luồng lưu file từ tin nhắn

1. Người dùng nhấn vào nút **"Lưu"** tại một tin nhắn chứa tệp.
2. SaveFileAsync() được gọi, truyền vào đối tượng ChatMessage.
3. Hệ thống kiểm tra:
    - Nếu fileMessage == null hoặc không phải tệp (IsFile == false) → **thoát**.
4. Mở hộp thoại SaveFileDialog, gợi ý tên file từ fileMessage.FileName.
5. Nếu người dùng **xác nhận**:
    - Giải mã FileContentBase64 thành mảng byte.
    - Ghi nội dung xuống file tại đường dẫn người dùng chọn.
    - Hiển thị thông báo "Lưu thành công".
6. Nếu lỗi xảy ra trong quá trình giải mã hoặc ghi file:
    - Hiển thị hộp thoại lỗi tương ứng.

2.3.6. Luồng xuất file lịch sử trò chuyện

1. Người dùng nhấn nút **Xuất lịch sử** trong giao diện.
2. ExportHistoryCommand gọi ExportChatHistory().
3. Hệ thống mở hộp thoại chọn nơi lưu file (SaveFileDialog).
4. Nếu người dùng xác nhận, chương trình duyệt toàn bộ danh sách Messages.
5. Mỗi tin nhắn được định dạng và ghi vào file .txt.
6. Sau khi ghi xong, hệ thống hiển thị thông báo "Lưu thành công".

2.4. Thiết kế giao thức truyền thông WebSocket

Trong ứng dụng chat, WebSocket được sử dụng để duy trì kết nối thời gian thực giữa Client (ứng dụng WPF) và Server, đảm bảo thông tin nhắn được truyền đi ngay lập tức mà không cần tạo lại kết nối mới như HTTP truyền thống.

Giao thức truyền thông được thiết kế dựa trên các bản tin JSON, mỗi bản tin có trường Type để xác định mục đích của tin nhắn, và các trường dữ liệu phụ thuộc vào loại tin nhắn.

2.4.1. Cấu trúc bản tin cơ bản (ChatMessageBase)

Mọi bản tin đều kế thừa từ lớp ChatMessageBase và bắt buộc phải có trường Type để xác định loại tin nhắn.

Program.cs

```
public class ChatMessageBase
{
    public string Type { get; set; }
}
```

2.4.2. Các bản tin từ Client lên Server và từ Server xuống Client

Để đảm bảo giao tiếp hiệu quả và rõ ràng giữa Client và Server, chúng tôi đã định nghĩa một tập hợp các bản tin JSON. Mỗi bản tin đều có trường Type để xác định mục đích của nó, và các trường dữ liệu bổ sung tùy thuộc vào loại bản tin.

WpfChatApp.sln

```
using System.Text.Json.Serialization;

// Lớp cơ sở cho mọi tin nhắn
public class ChatMessageBase
{
    public string Type { get; set; }
}

// --- CÁC MESSAGE MỚI ---

//Cilent gửi khi đăng nhập hệ thống
public class LoginMessage : ChatMessageBase
{
    public string Username { get; set; }
}

// Client gửi khi tham gia phòng chat cụ thể
public class JoinRoomMessage : ChatMessageBase
{
    public string RoomName { get; set; }
}

// Client gửi lên khi muốn gửi tin nhắn văn bản tới phòng hoặc một cá nhân
public class ChatTextMessage : ChatMessageBase
{
    public string Sender { get; set; }
    public string Receiver { get; set; } // Có thể null nếu gửi vào phòng
    public string Room { get; set; }
    public string Content { get; set; }
    public string Timestamp { get; set; }
}

//Cilent gửi khi đăng xuất hệ thống
public class LogoutMessage : ChatMessageBase
{
    public string Username { get; set; }
}

// --- XỬ LÝ GỬI VÀ NHẬN TIN NHẮN TRÊN SERVER  ---


// Server thông báo thành viên tham gia
```

```
public class UserJoined : ChatMessageBase
{
    public string Username { get; set; }
}

//Server thông báo thành viên rời đi
public class UserLeft : ChatMessageBase
{
    public string Username { get; set; }
}

// Nhận message từ client
public class IncomingTextMessage : ChatMessageBase
{
    public string Sender { get; set; }
    public string Room { get; set; }
    public string Content { get; set; }
    public string Timestamp { get; set; }
}

//Server thông báo lỗi tới Cilent
public class ErrorMessage : ChatMessageBase
{
    public string Message { get; set; }
}
```

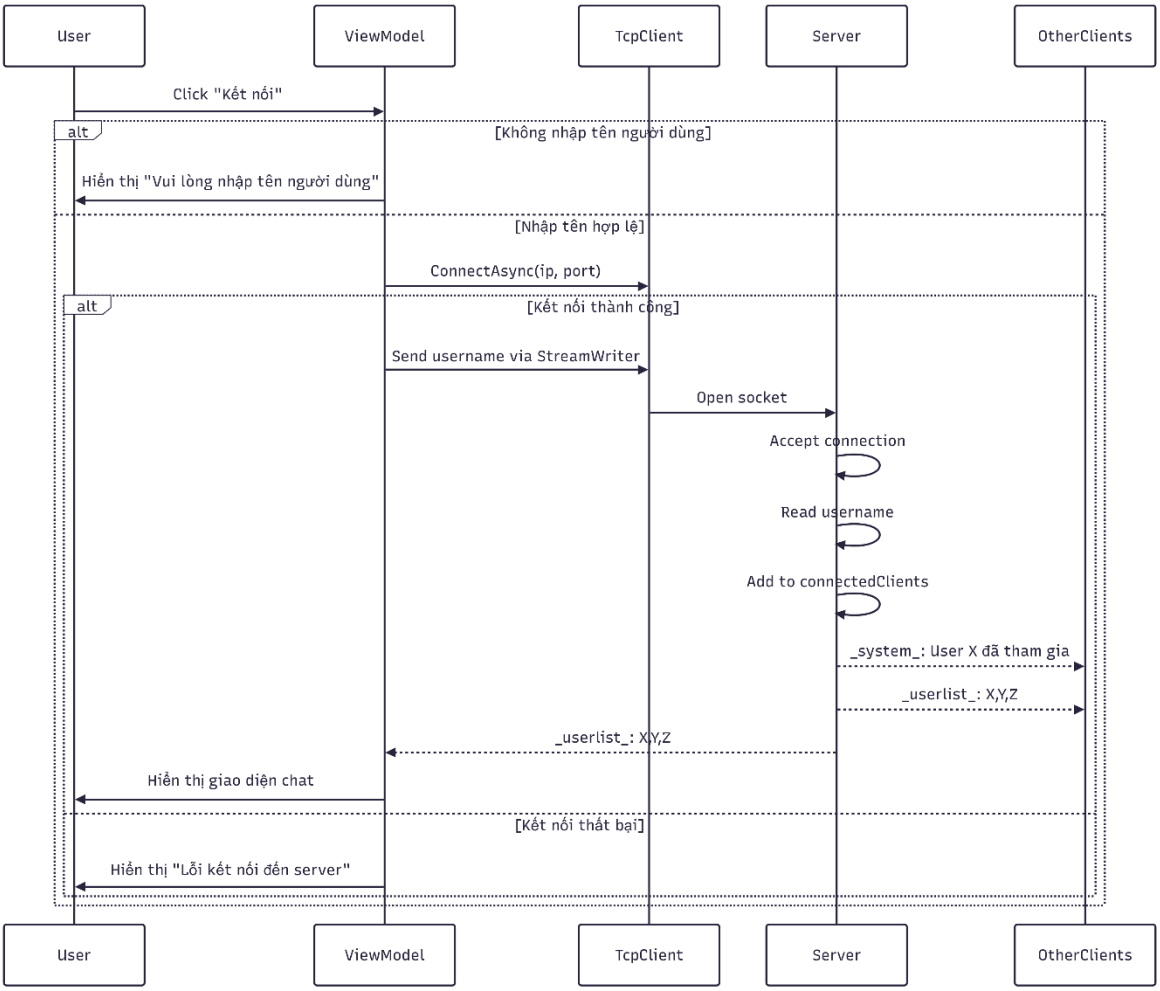
Giải thích chi tiết về các loại bản tin:

+ *LoginMessage*: Client gửi khi đăng nhập hệ thống
+ *JoinRoomMessage*: Client thông báo tham gia kênh chat với tên người dùng.
+ *ChatTextMessage*: Client gửi tin nhắn mới
+ *LogoutMessage*: Client gửi khi đăng xuất hệ thống
+ *UserJoined*: Server thông báo thành viên tham gia.
+ *UserLeft*: Server thông báo thành viên rời đi.
+ *IncomingTextMessage*: Server nhận tin nhắn từ cilent
+ *ErrorMessage*: Server thông báo lỗi tới cilent

2.5. Lưu đồ thuật toán các chức năng chính

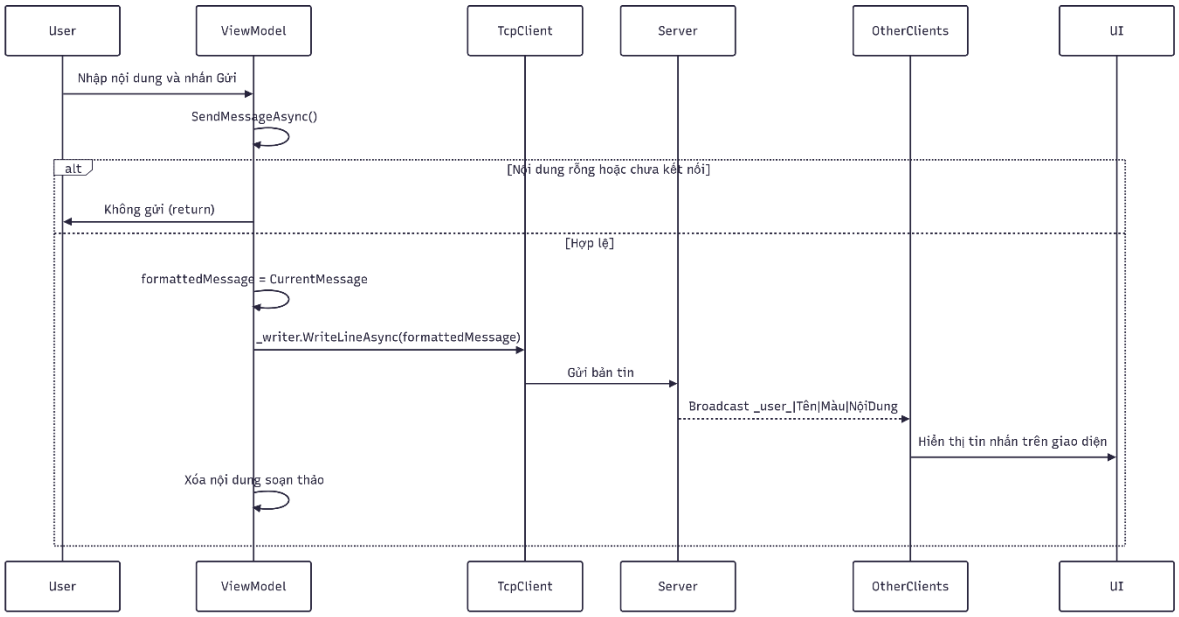
*** Lưu ý: "OtherClients" ở đây nghĩa là các người dùng khác!**

2.5.1. Lưu đồ tạo/tham gia phòng chat



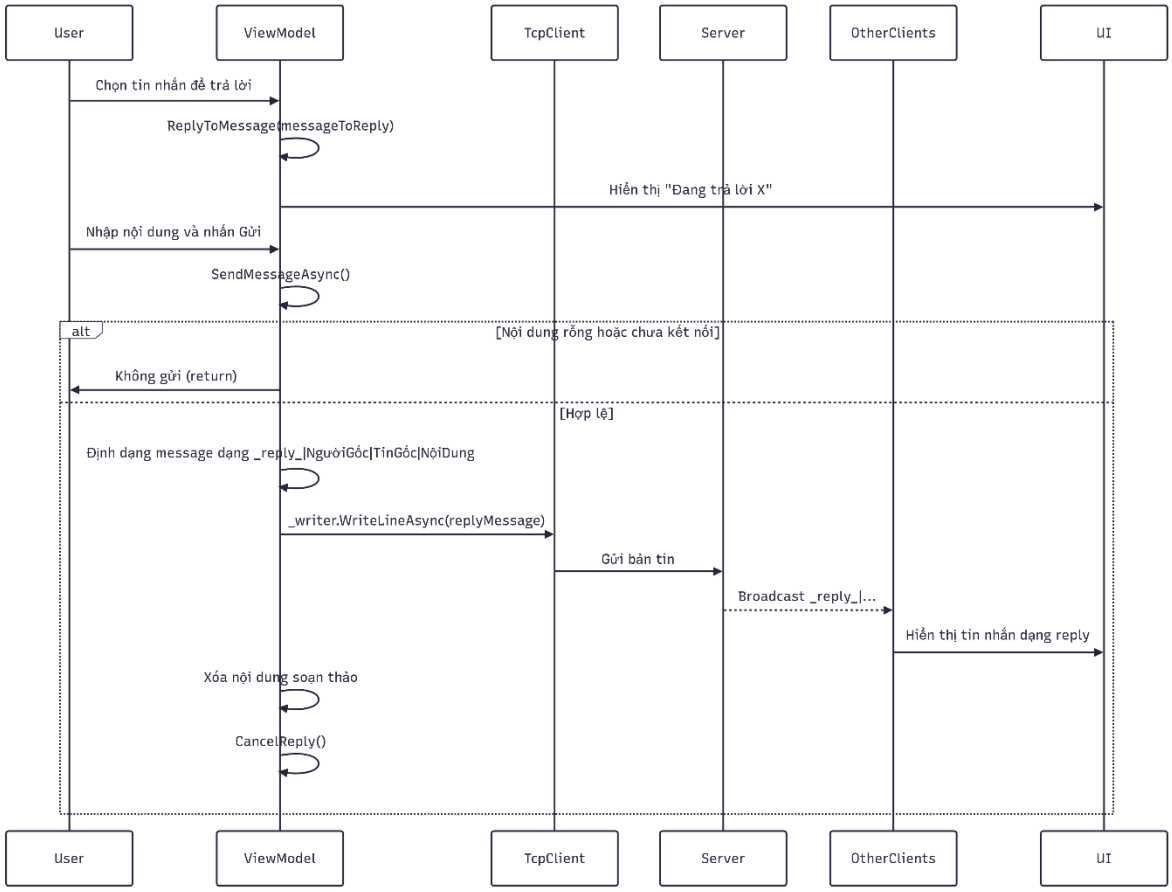
Hình 1. Lưu đồ tạo/tham gia phòng chat

2.5.2. Lưu đồ gửi tin nhắn thường



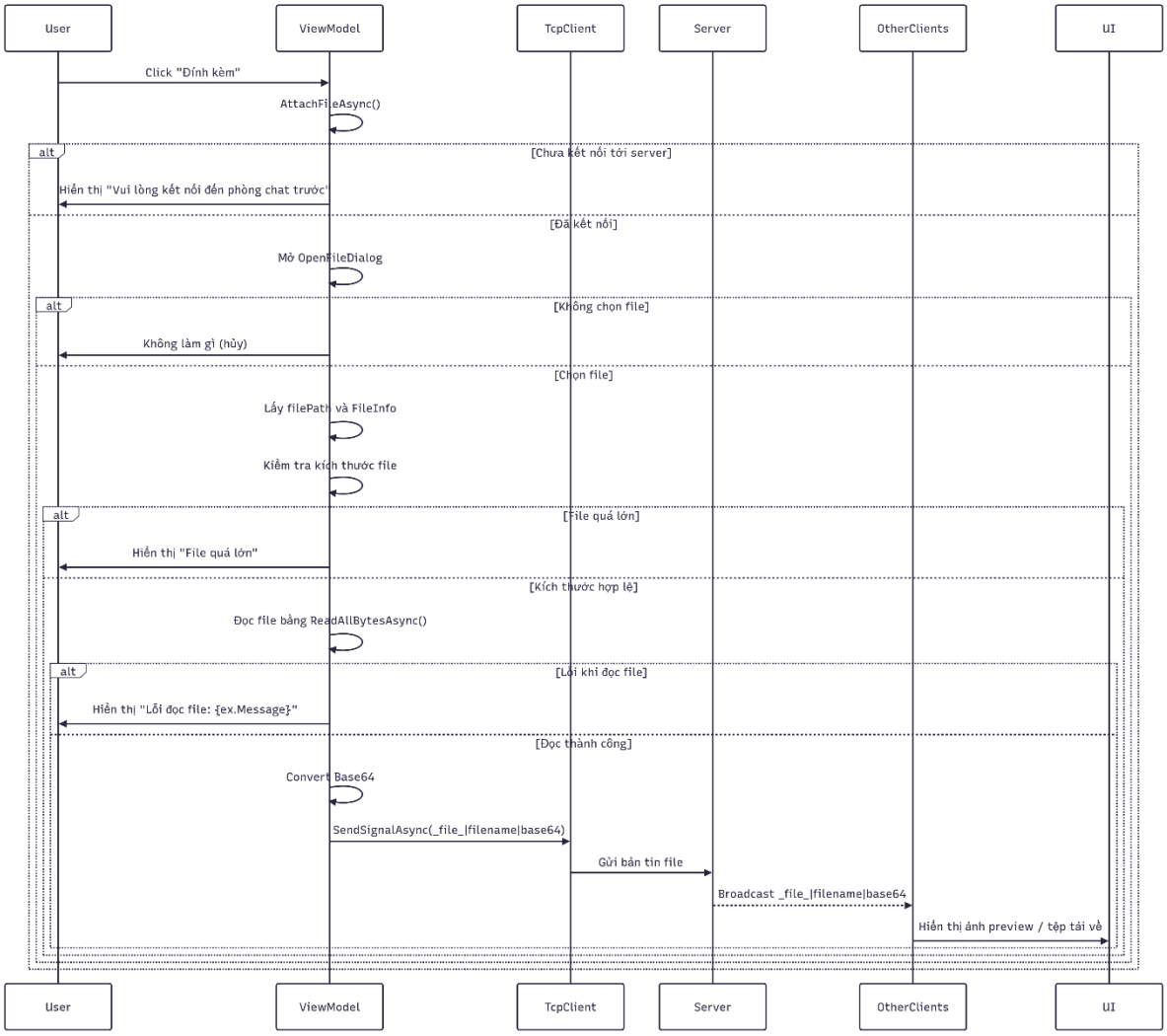
Hình 2. Lưu đồ gửi tin nhắn thường

2.5.3. Lưu đồ trả lời tin nhắn



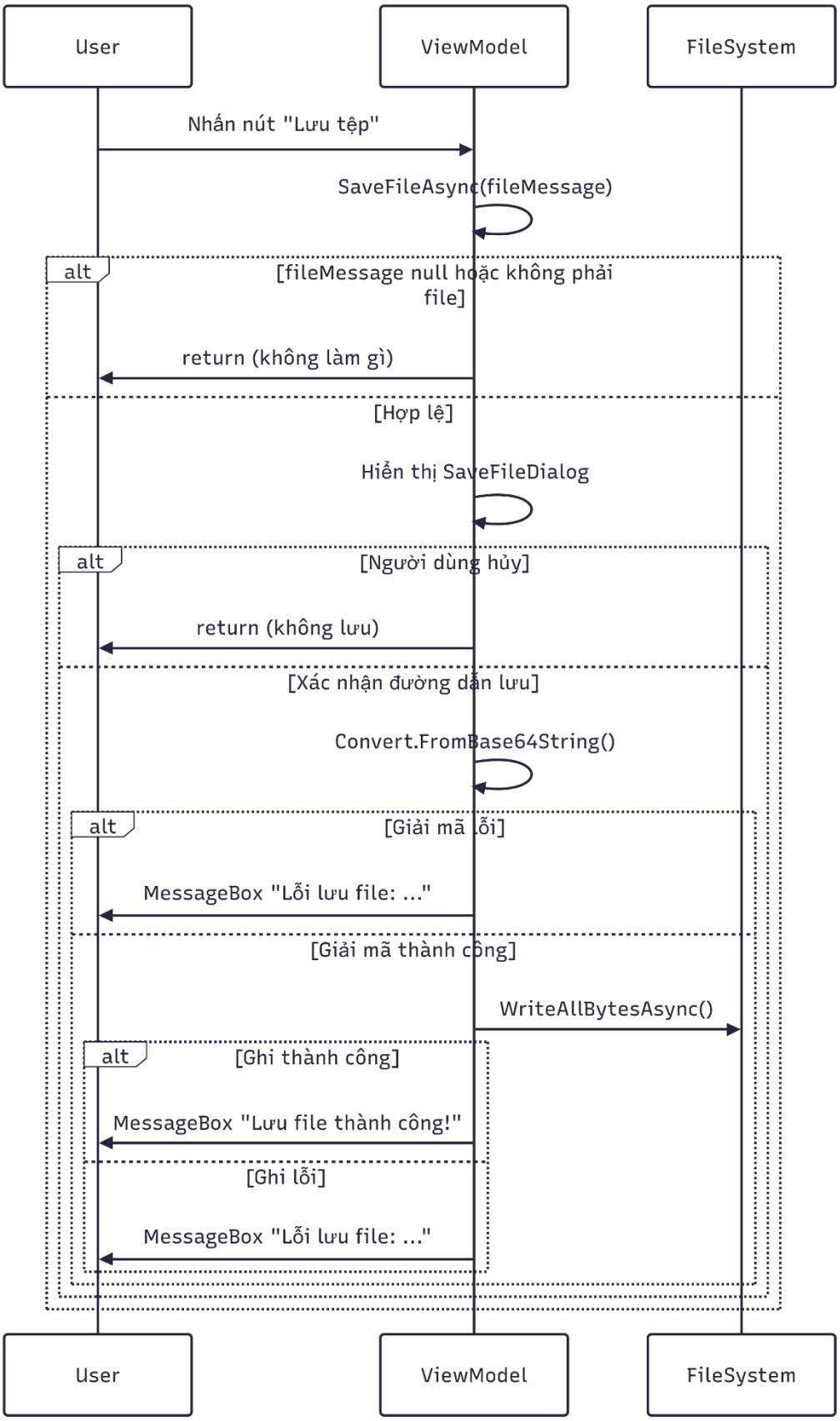
Hình 3. Lưu đồ gửi trả lời tin nhắn

2.5.4. Lưu đồ gửi tệp tin



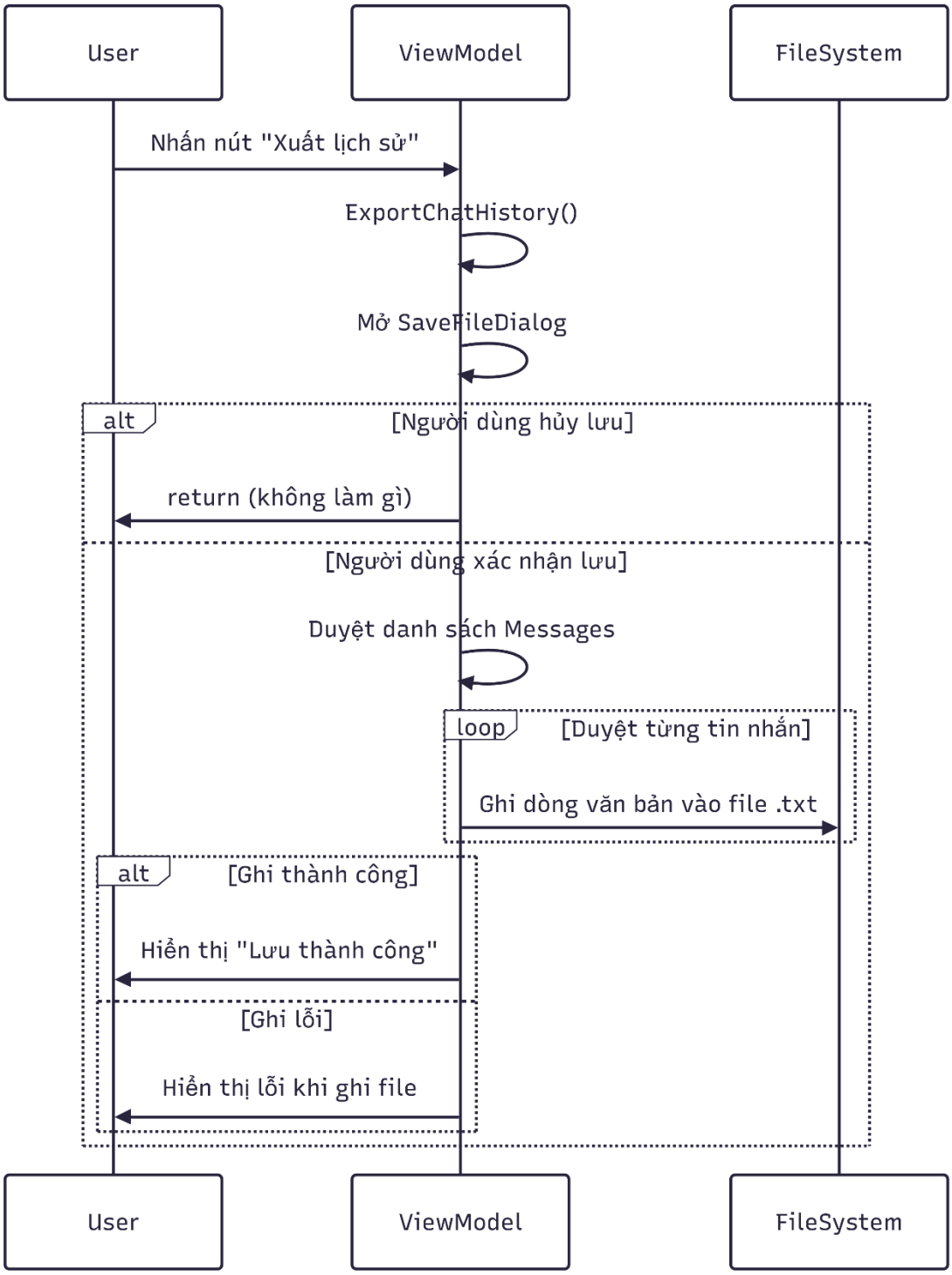
Hình 4. Lưu đồ gửi tệp tin

2.5.5. Lưu đồ lưu file từ tin nhắn


User
ViewModel
FileSystem
Nhấn nút "Lưu tệp"
SaveFileAsync(fileMessage)
alt
[fileMessage null hoặc không phải file]
return (không làm gì)
[Hợp lệ]
Hiển thị SaveFileDialog
alt
[Người dùng hủy]
return (không lưu)
[Xác nhận đường dẫn lưu]
Convert.FromBase64String()
alt
[Giải mã lỗi]
MessageBox "Lỗi lưu file: ..."
[Giải mã thành công]
WriteAllBytesAsync()
alt
[Ghi thành công]
MessageBox "Lưu file thành công!"
[Ghi lỗi]
MessageBox "Lỗi lưu file: ..."
User
ViewModel
FileSystem

Hình 5. Lưu đồ lưu tệp tin

2.5.6. Lưu đồ xuất lịch sử trò chuyện



Hình 6. Lưu đồ xuất lịch sử trò chuyện

CHƯƠNG 3: TRIỂN KHAI HỆ THỐNG

3.1. Cấu trúc project

- **Client (WPF - WpfChatApp Project)**
  - **Models/:** Chứa các lớp mô hình dữ liệu và logic nghiệp vụ chính của ứng dụng chat. Đây là nơi định nghĩa cấu trúc dữ liệu và quy tắc kinh doanh.
    - `ClientInfo.cs`
    - `Message.cs` (và các lớp tin nhắn kế thừa)
    - `ChatClient.cs` (có thể bao gồm logic kết nối và xử lý tin nhắn phía client)
  - **Views/:** Chứa các file XAML định nghĩa giao diện người dùng. Các View này sẽ liên kết (bind) với các ViewModel.
    - `MainWindow.xaml` (Giao diện chính của ứng dụng chat)
    - `MessageTemplate.xaml` (Nếu có các template riêng cho tin nhắn)
  - **ViewModels/:** Chứa các lớp ViewModel, chịu trách nhiệm xử lý logic cho View, quản lý trạng thái của View và cung cấp dữ liệu cho View thông qua Data Binding.
    - `MainWindowViewModel.cs` (ViewModel chính cho MainWindow)
    - `BaseViewModel.cs` (Lớp cơ sở cho các ViewModel, có thể triển khai `INotifyPropertyChanged`)
  - **Services/:** Có thể chứa các dịch vụ (ví dụ: `NetworkService.cs` để quản lý kết nối mạng, `FileService.cs` để xử lý tệp).
  - **Properties/:** Chứa cấu hình dự án, bao gồm `launchSettings.json` cho cấu hình khởi chạy.
  - **App.xaml, App.xaml.cs:** File cấu hình và khởi tạo ứng dụng WPF.
- **Server (.NET Console App - ChatServer Project)**
  - **Program.cs:** File cấu hình và khởi tạo ứng dụng Server, bao gồm thiết lập lắng nghe kết nối TCP và quản lý các client đang kết nối.
  - **Models/:** Có thể chứa các lớp mô hình dữ liệu liên quan đến server (ví dụ: `ClientInfo.cs` được chia sẻ hoặc định nghĩa riêng cho server).
    - `ClientInfo.cs`
  - **Properties/:** Chứa cấu hình dự án của Server.
- **Các file cấu hình chung:**
  - `appsettings.json`, `appsettings.Development.json`: Các file cấu hình ứng dụng (nếu được sử dụng cho server hoặc client).

3.2. Triển khai phía Server (.NET Console App)

3.2.1. Khởi tạo TCP Listener và Xử lý kết nối

Trong dự án Server (.NET Console App), việc khởi tạo và quản lý kết nối được thực hiện thông qua **TcpListener.**

- **Program.cs:** Là nơi chính thiết lập và quản lý server.
  - **Khởi tạo `TcpListener`:** Server mở cổng 8888 và bắt đầu lắng nghe kết nối từ các client.
  - **Xử lý kết nối mới:** Khi một client kết nối thành công, server tạo một luồng xử lý riêng biệt cho client đó để đọc và ghi dữ liệu.
  - **Vòng lặp nhận/gửi tin nhắn:** Luồng xử lý duy trì một vòng lặp để liên tục nhận tin nhắn từ client và gửi phản hồi.
  - **Ngắt kết nối:** Khi client đóng kết nối hoặc xảy ra lỗi, server sẽ phát hiện và thực hiện các bước dọn dẹp (ví dụ: xóa client khỏi danh sách `connectedClients`).
- **Cấu hình quan trọng:**
  - Dữ liệu được đọc và ghi qua `NetworkStream` của `TcpClient`.
  - Việc tuần tự hóa và giải tuần tự hóa tin nhắn văn bản được thực hiện thủ công dựa trên giao thức đã định nghĩa.

3.2.2. Quản lý trạng thái chat (Program.cs & ClientInfo) Việc quản lý trạng thái của các phiên làm việc và thông tin client được xử lý trực tiếp trong

`Program.cs` thông qua cấu trúc dữ liệu trong bộ nhớ.

- **`Program.cs`:** Đóng vai trò trung tâm điều phối và quản lý toàn bộ luồng chat giữa các người chơi.
  - `static readonly Dictionary<TcpClient, ClientInfo> connectedClients`: Là một `Dictionary` chứa thông tin của tất cả các client đang kết nối.
    - **Khóa (`TcpClient`):** Đối tượng kết nối mạng duy nhất cho mỗi client.
    - **Giá trị (`ClientInfo`):** Một lớp chứa các thông tin của client đó.
  - **`ClientInfo.cs`:** Đại diện cho mỗi người dùng đang kết nối.
    - **Thuộc tính:** `Username`, `ColorHex`, và đối tượng `StreamWriter` để gửi tin nhắn.
  - **Chức năng chính:**
    - **Thêm/Xóa Client:** Khi client kết nối, thông tin của họ được thêm vào `connectedClients` và một màu sắc được gán. Khi client ngắt kết nối, họ sẽ bị xóa khỏi danh sách.
    - **Phân phối tin nhắn:** Server nhận tin nhắn từ một client, định dạng lại và **broadcast** (phân phối) cho tất cả các client khác trong `connectedClients`.
    - **Gửi thông báo hệ thống:** Gửi các thông báo như "user has joined" hoặc "user has left" cho tất cả mọi người.

3.2.3. Xử lý logic thông điệp và giao tiếp

Toàn bộ giao tiếp giữa client và server đều dựa trên một giao thức truyền thông dựa trên văn bản tự định nghĩa, với các thông điệp được phân tách bằng ký tự " | ". Server chịu trách nhiệm phân tích và xử lý các loại tin nhắn khác nhau.

- **Phân tích và Xử lý tin nhắn:**
  - Server đọc luồng dữ liệu từ client và phân tích chuỗi tin nhắn nhận được.
  - Dựa vào tiền tố của tin nhắn (ví dụ: `_user_`, `_reply_`, `_file_`), server xác định loại tin nhắn và thực hiện các hành động tương ứng:
    - **Đăng ký Nickname:** Tin nhắn đầu tiên client gửi để đăng ký tên.
    - **Tin nhắn thường:** Nhận nội dung, định dạng lại và phát đi (broadcast) cho tất cả người dùng.
    - **Trả lời tin nhắn:** Xử lý tin nhắn trả lời và phát đi với thông tin người trả lời và nội dung gốc.
    - **Gửi File:** Nhận tên file và nội dung được mã hóa Base64, sau đó phát đi cho tất cả client.
  - **Gửi phản hồi:** Tin nhắn phản hồi hoặc tin nhắn broadcast được gửi lại cho từng người chơi thông qua `StreamWriter` của `ClientInfo`.
- **Các loại tín hiệu (Giao thức truyền thông):**
  - **Đăng ký:** `Nickname` (Client Gửi đi)
  - **Tin nhắn thường:** `Nội dung tin nhắn` (Client Gửi đi) -> `_user_|SenderName|SenderColor|Content` (Server Phát đi)
  - **Trả lời:** `_reply_|RepliedToUser|RepliedToContent|NewContent` (Client Gửi đi) -> `_reply_|SenderName|SenderColor|RepliedToUser|RepliedToContent|NewContent` (Server Phát đi)
  - **Gửi File:** `_file_|FileName|Base64Content` (Client Gửi đi) -> `_file_|SenderName|SenderColor|FileName|Base64Content` (Server Phát đi)
  - **Danh sách User:** (Client không gửi) -> `_userlist_|User1,User2,User3` (Server Phát đi)
  - **Hệ thống:** (Client không gửi) -> `_system_|Nội dung thông báo` (Server Phát đi)

Việc định nghĩa rõ từng loại tin nhắn và giao thức giúp server và client có thể hiểu được ý nghĩa của mỗi gói tin nhận được, giảm lỗi và dễ mở rộng.

3.3. Triển khai phía Client

Phía Client được phát triển bằng C# và WPF, tuân thủ mô hình kiến trúc MVVM để đảm bảo tính tổ chức, dễ bảo trì và mở rộng của mã nguồn.

3.3.1. Cấu trúc MVVM

- **View (MainWindow.xaml):** Định nghĩa giao diện người dùng chính của ứng dụng chat. Đây là nơi người dùng tương tác, bao gồm các thành phần như TextBox để nhập nickname, IP, Port, nút "Join Chat", khu vực hiển thị tin nhắn, danh sách người dùng online, v.v.. View không chứa logic xử lý nghiệp vụ mà chỉ tập trung vào việc trình bày dữ liệu và gửi các hành động của người dùng tới ViewModel thông qua Data Binding và Commands.
- **ViewModel (MainWindowViewModel.cs):** Là cầu nối giữa View và Model, chứa logic xử lý các hành động từ View và cung cấp dữ liệu cho View hiển thị.
    - **Data Binding:** Các thuộc tính trong ViewModel (ví dụ: `Nickname`, `IP`, `Port`, `ConnectionPanelVisibility`, `OnlineUsers`, `Messages`, `ReplyPreview`) được liên kết hai chiều với các điều khiển trên View. Khi thuộc tính của ViewModel thay đổi, View tự động cập nhật, và ngược lại.
    - **Commands:** Xử lý các hành động từ người dùng (ví dụ: `ConnectCommand` khi nhấn "Join Chat", `SendCommand` khi gửi tin nhắn, `ReplyCommand` khi chọn trả lời tin nhắn). Command sẽ thực thi các phương thức tương ứng trong ViewModel để xử lý nghiệp vụ.
    - **Quản lý trạng thái UI:** ViewModel kiểm soát việc hiển thị/ẩn các panel UI (ví dụ: `ConnectionPanelVisibility` để ẩn panel kết nối và hiển thị giao diện chat chính).
- **Model (ClientInfo.cs, Message.cs, ChatClient.cs):** Đại diện cho dữ liệu và các quy tắc nghiệp vụ.
    - `ClientInfo.cs`: Chứa thông tin của một người dùng chat (nickname, màu sắc).
    - `Message.cs`: Định nghĩa cấu trúc của một tin nhắn chat, bao gồm nội dung, người gửi, loại tin nhắn (thường, trả lời, file), v.v..
    - `ChatClient.cs` (hoặc logic tương tự trong ViewModel): Xử lý việc thiết lập kết nối TCP đến Server, gửi nhận dữ liệu và giải mã/mã hóa các tin nhắn theo giao thức đã định nghĩa.

3.3.2. Luồng hoạt động chính của Client

- **Khởi động ứng dụng:** Người dùng chạy `WpfChatApp.exe`.

`MainWindow` (View) được hiển thị, và `MainWindowViewModel` được khởi tạo và gán làm `DataContext` cho View.

- **Kết nối Server:**
    - Người dùng nhập Nickname, IP, Port vào các TextBox trên View. Các giá trị này được cập nhật vào thuộc tính tương ứng trong `MainWindowViewModel` thông qua Data Binding.
    - Người dùng nhấn nút "Join Chat". View gọi đến `ConnectCommand` trong ViewModel.
    - Phương thức `ConnectAsync` trong ViewModel được thực thi, tạo kết nối đến Server và gửi Nickname.

    - Khi kết nối thành công, ViewModel cập nhật thuộc tính `ConnectionPanelVisibility` thành `Collapsed`, khiến View tự động ẩn panel kết nối và hiển thị giao diện chat chính. Danh sách người dùng online cũng được cập nhật vào `OnlineUsers collection` trong ViewModel và được View hiển thị.
- **Gửi/Nhận tin nhắn:**
    - Khi người dùng nhập tin nhắn và nhấn gửi, View gọi đến `SendCommand` trong ViewModel. ViewModel kiểm tra loại tin nhắn (thường, trả lời, file) và tạo chuỗi tín hiệu phù hợp theo giao thức.
    - Tin nhắn được gửi lên Server.
    - Khi Server gửi tin nhắn về, ViewModel nhận được, xử lý và thêm tin nhắn vào `Messages collection`. View tự động cập nhật để hiển thị tin nhắn mới.
- **Trả lời tin nhắn:**
    - Người dùng chuột phải vào một tin nhắn và chọn "Trả lời". View thực thi `ReplyCommand` với tham số là tin nhắn được chọn.
    - ViewModel lưu tin nhắn đó vào một thuộc tính tạm thời (

`_replyingToMessage`) và cập nhật các thuộc tính liên quan đến xem trước trả lời (`ReplyPreview...`).

    - View tự động hiển thị `ReplyPreviewPanel`.
    - Khi người dùng nhập nội dung trả lời và nhấn gửi, `SendCommand` sẽ kiểm tra `_replyingToMessage` để tạo chuỗi tín hiệu trả lời phù hợp.
- **Ngắt kết nối:**
    - Khi một client đóng cửa sổ ứng dụng, sự kiện

`Window_Closing` trong `MainWindow.xaml.cs` (code-behind) sẽ gọi đến hàm `DisconnectAndCleanup` trong ViewModel.

    - ViewModel đóng kết nối với Server và đặt lại các thuộc tính về trạng thái ban đầu. View tự động quay về màn hình kết nối.

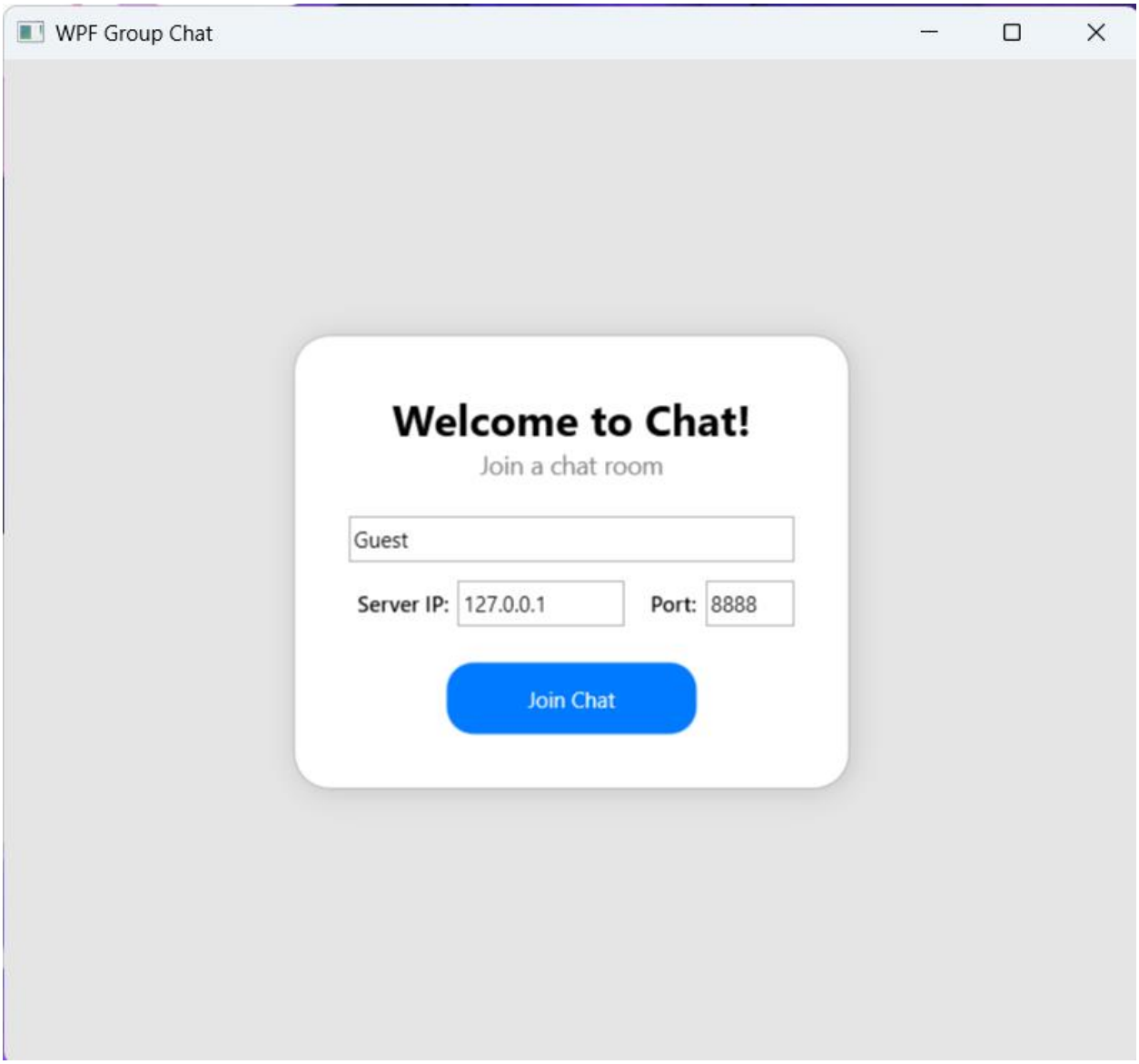
3.3.3. Các đối tượng tin nhắn (Message Classes)

Toàn bộ giao tiếp giữa client và server đều dựa trên các tin nhắn văn bản được định dạng theo giao thức tự định nghĩa. Phía client có các lớp hoặc cấu trúc dữ liệu tương ứng để:

- **Tạo tin nhắn:** Mã hóa dữ liệu người dùng nhập thành chuỗi tin nhắn phù hợp với giao thức trước khi gửi lên Server (ví dụ: thêm tiền tố `_reply_` hoặc `_file_`).
- **Phân tích và hiển thị tin nhắn:** Giải mã chuỗi tin nhắn nhận được từ Server để trích xuất thông tin (người gửi, nội dung, màu sắc, tên file) và tạo các đối tượng `Message` thích hợp để thêm vào danh sách hiển thị trên View.
- Việc tuân thủ giao thức giúp client và server hiểu rõ ý nghĩa của mỗi gói tin, đảm bảo giao tiếp rõ ràng và nhất quán.

3.4. Giao diện người dùng

- Màn hình Lobby:
  + Nút "Join Chat" để tạo phòng hoặc vào phòng.
  + Form "Guest" với input để nhập tên
  + Form "Server IP" với input để nhập IP máy chủ muốn kết nối tới
  + Form “Port” với input để nhập cổng vào (hiện tại mặc định là 8888)



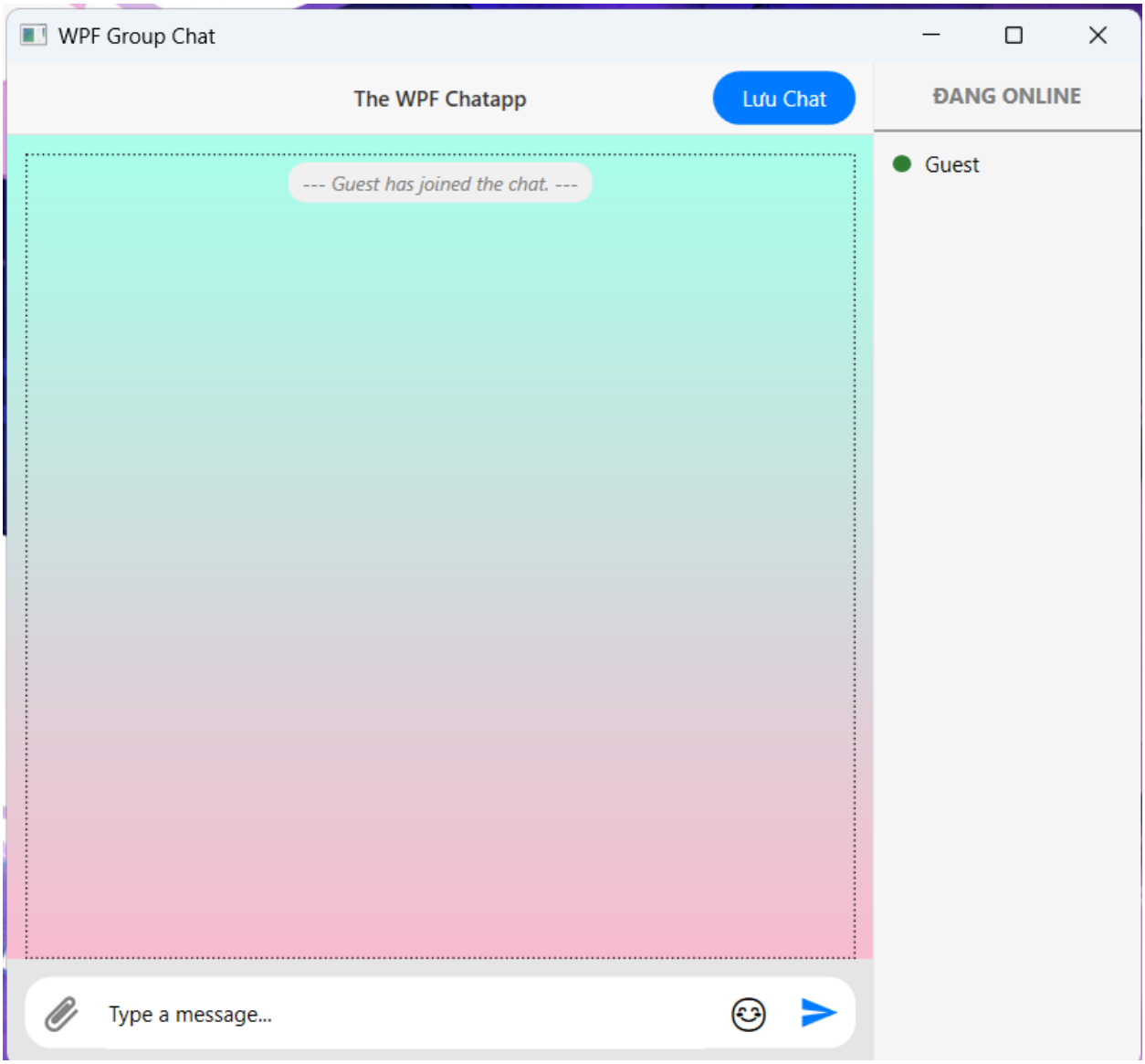
Hình 3.1: Màn hình Lobby

- Màn hình Phòng chat:
  + Nút "Lưu chat" để lưu lại đoạn chat
  + Form “Type a message” với input để nhập tin nhắn
  + Nút " " để tìm file muốn gửi lên đoạn chat
  + Nút " " để thả emoji lên đoạn chat

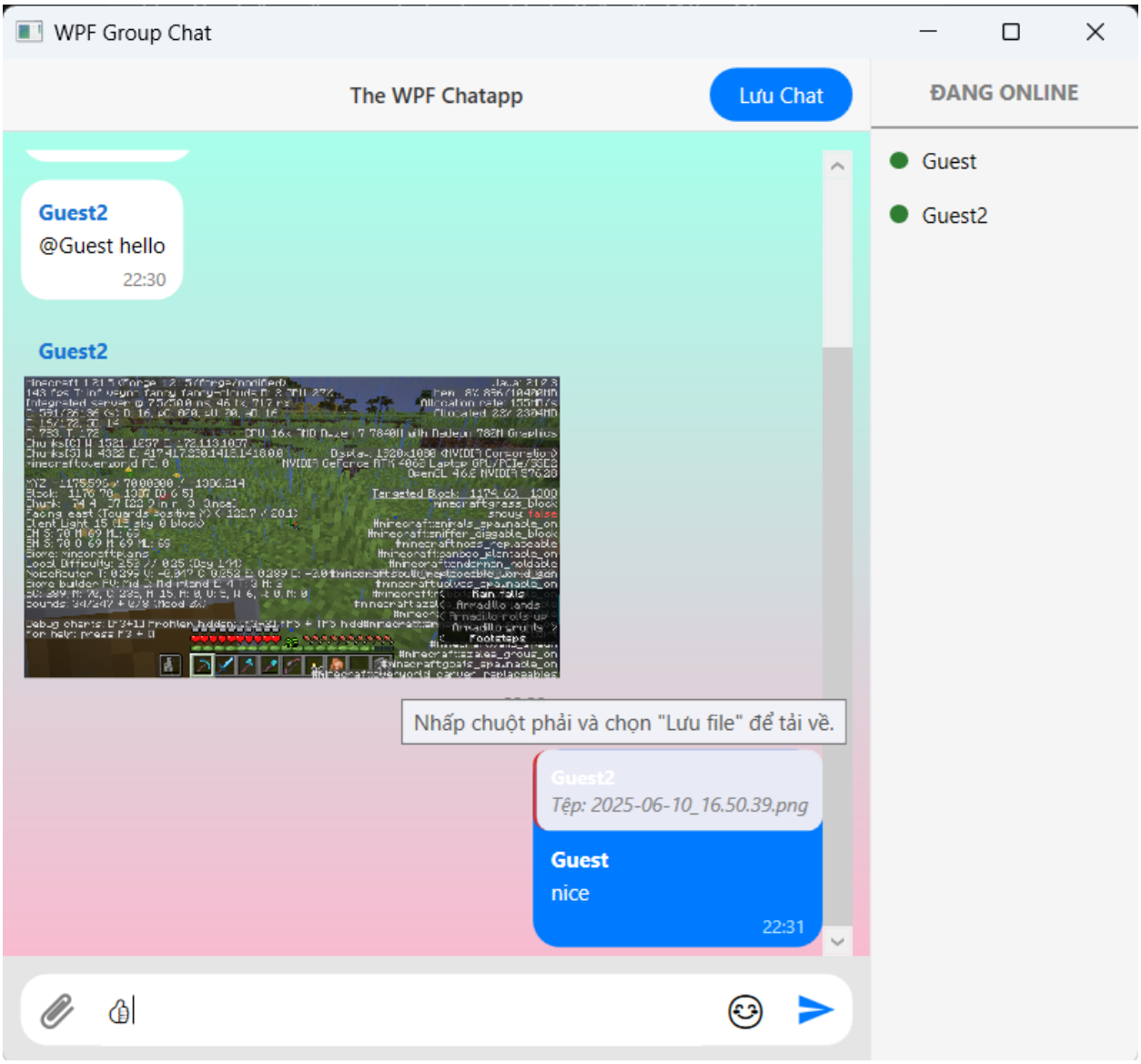
+ Nút " " để gửi tin nhắn
+ Bảng “Đang online” thông báo các cilent trong đoạn chat

-Một số chức năng khác:

+ Click chuột phải vào file, ảnh và video để lưu file hoặc trả lời
+ Click chuột trái vào cilent trong bảng “Đang online” để gửi tin nhắn riêng



Hình 3.2: Màn hình Phòng chat của người dùng



Hình 3.3: Giao diện người dùng khi đang sử dụng

KẾT LUẬN

1. Kết quả đạt được

Sau quá trình tìm hiểu và phát triển, nhóm đã xây dựng thành công ứng dụng chat thời gian thực, bao gồm:

- Server: Sử dụng WebSocket để duy trì kết nối song công với nhiều client, cho phép nhận và phân phối tin nhắn nhanh chóng.
- Client (ứng dụng WPF): Giao diện người dùng thân thiện, cho phép gửi và nhận tin nhắn, hiển thị thông tin cuộc trò chuyện theo thời gian thực.

Nhờ việc thiết kế giao thức truyền thông WebSocket rõ ràng và phân tách thành các bản tin (ví dụ: SendMessage, ReceiveMessage, UserJoined, UserLeft…), ứng dụng đạt được:

- Hiệu năng ổn định, độ trễ thấp.
- Dễ dàng mở rộng thêm tính năng mới trong tương lai.
- Mã nguồn có cấu trúc rõ ràng, dễ bảo trì.

Bên cạnh đó, nhóm đã trau dồi và củng cố các kiến thức về lập trình mạng, kỹ năng thiết kế phần mềm, làm việc nhóm và viết báo cáo kỹ thuật.

2. Hướng phát triển

Mặc dù ứng dụng hiện tại đã đáp ứng yêu cầu cơ bản của đề tài, nhóm nhận thấy còn nhiều điểm có thể cải thiện và phát triển thêm, cụ thể:

+ Bổ sung chức năng **chat nhóm**, hỗ trợ nhiều phòng chat song song.
+ Xây dựng **hệ thống đăng nhập, quản lý tài khoản** để tăng tính cá nhân hóa.
+ Hỗ trợ **gửi và nhận file**, emoji, sticker để làm phong phú trải nghiệm người dùng.
+ Nâng cấp giao diện đồ họa theo xu hướng hiện đại (Material Design).
+ Triển khai server lên nền tảng cloud để có thể sử dụng trên nhiều thiết bị thật.

Nhóm hy vọng trong các phiên bản tiếp theo, sản phẩm sẽ hoàn thiện hơn và tiệm cận nhu cầu thực tế.

TÀI LIỆU THAM KHẢO

[1] Microsoft Docs – *ASP.NET Core SignalR* https://learn.microsoft.com/en-us/aspnet/core/signalr/introduction

[2] Microsoft Docs – *WebSockets in .NET* https://learn.microsoft.com/en-us/dotnet/api/system.net.websockets

[3] Microsoft Docs – *Create a WPF app* https://learn.microsoft.com/en-us/visualstudio/get-started/csharp/tutorial-wpf

[4] WebSocket.org – *WebSocket Protocol* https://www.websocket.org/aboutwebsocket.html

[5] Json.NET – *Popular high-performance JSON framework for .NET* https://www.newtonsoft.com/json