**ĐẠI HỌC QUỐC GIA TP.HCM**

**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN**

**KHOA ĐIỆN TỬ - VIỄN THÔNG**

**BÁO CÁO ĐỒ ÁN CUỐI KỲ**

**Môn: Lập trình Java**

**Lớp: 14DTV2**

***Đề tài: CHƯƠNG TRÌNH LAN CHAT***

**GVHD: Lê Viết Long**

**Nhóm thực hiện:**

1. **Nguyễn Minh Trí 1420142**
2. **Nguyễn Bảo Trung 1420144**
3. **Phan Việt Trung 1420145**
4. **Dương Nhật Trường 1420261**
5. **Lê Thanh Tùng 1420266**

**Tp.HCM, ngày 20 tháng 12 năm 2017**

**MỤC LỤC**

[**LỜI NÓI ĐẦU** 3](#_Toc501539309)

[**CHƯƠNG 1: TỔNG QUAN** 4](#_Toc501539310)

[**1.1** **Tổng quan về vấn đề được nghiên cứu** 4](#_Toc501539311)

[**1.2** **Nhiệm vụ đồ án** 4](#_Toc501539312)

[**1.3** **Cấu trúc đồ án** 4](#_Toc501539313)

[**CHƯƠNG 2: CƠ SỞ LÝ THUYẾT** 5](#_Toc501539314)

[**2.1** **Khái niệm địa chỉ và cổng** 5](#_Toc501539315)

[**2.2** **Sơ lược về lập trình đa luồng** 5](#_Toc501539316)

[***2.2.1*** ***Khái niệm luồng (Thread)*** 5](#_Toc501539317)

[***2.2.2 Đồng bộ hóa*** 6](#_Toc501539318)

[***2.2.3*** ***Deadlock*** 7](#_Toc501539319)

[**CHƯƠNG 3: PHÂN TÍCH VÀ THIẾT KẾ YÊU CẦU HỆ THỐNG** 8](#_Toc501539320)

[**3.1** **Phân tích hệ thống** 8](#_Toc501539321)

[***3.1.1*** ***Phân tích nhu cầu thực tiễn*** 8](#_Toc501539322)

[***3.1.2*** ***Yêu cầu đề ra*** 8](#_Toc501539323)

[3.1.3 ***Mục tiêu*** 9](#_Toc501539324)

[**3.2** **Thiết kế hệ thống** 9](#_Toc501539325)

[***3.2.1*** ***Mô hình Client – Server*** 9](#_Toc501539326)

[***3.2.2*** ***Sơ đồ Usecase tổng thể*** 10](#_Toc501539327)

[***3.2.3*** ***Sơ đồ hoạt động*** 11](#_Toc501539328)

[**3.3** **Mô tả các Class & Object sử dụng trong đồ án** 15](#_Toc501539329)

[***3.3.1*** ***Client*** 15](#_Toc501539330)

[***3.3.2*** ***Server*** 18](#_Toc501539331)

[**3.4** **Các hàm xử lý và ý nghĩa của hàm** 21](#_Toc501539333)

[***3.4.1*** ***Client*** 21](#_Toc501539334)

[***3.4.2*** ***Server*** 25](#_Toc501539335)

[**3.5** **Sơ đồ gọi hàm khi thực hiện chức năng** 32](#_Toc501539336)

[***3.5.1*** ***Đăng ký*** 32](#_Toc501539337)

[***3.5.2*** ***Đăng nhập*** 33](#_Toc501539339)

[***3.5.3*** ***Gửi tin nhắn đến Client*** 33](#_Toc501539341)

[***3.5.4*** ***Gửi tin nhắn đến Group*** 34](#_Toc501539345)

[***3.5.5*** ***Tạo Group*** 34](#_Toc501539348)

[***3.5.6*** ***Gửi file*** 35](#_Toc501539349)

[**CHƯƠNG 4: TỔNG KẾT VÀ KIẾN NGHỊ** 36](#_Toc501539352)

[**4.1 Kết quả đạt được** 36](#_Toc501539353)

[**4.2 Hạn chế** 36](#_Toc501539354)

[**4.3 Hướng phát triển** 36](#_Toc501539355)

[**4.4 Tài liệu tham khảo** 36](#_Toc501539356)

[**PHÂN CÔNG CÔNG VIỆC** 37](#_Toc501539357)

[**PHỤ LỤC – GIAO DIỆN MÀN HÌNH PHẦN MỀM DEMO CHAT** 38](#_Toc501539358)

**LỜI NÓI ĐẦU**

Ngày nay, cùng với sự phát triển của đất nước, các ngành khoa học, ngành công nghệ thông tin, đặc biệt là lĩnh vực mạng, internet đang trên đà phát triển với tốc độ rất lớn. Mạng máy tính ra đời đã giải quyết rất nhiều nhu cầu của con người…. trong đó có loại hình truyền thông qua mạng, con người có thể kết nối, nói chuyện gặp gỡ trên mọi nơi, mọi khoảng cách thông qua rất nhiều ứng dụng như Skype, Messenger, Zalo.

Do vậy, với mục tiêu mô phỏng mô hình này nhằm cung cấp một cái nhìn tổng quát nhất về đặc tính kỹ thuật, cách xây dựng một chương trình cung cấp khả năng giao tiếp qua mạng cho con người…. chúng em đã thực hiện đề tài: “Chương trình Lan Chat” bằng ngôn ngữ Java trên nền được viết bằng NetBeans IDE 8.2.

Do khá mới với Lập trình Java cũng như là đồ án lớn đầu tiên nên sẽ không tránh khỏi sai xót, rất mong nhận được sự góp ý của Thầy và các bạn.

Chúng em xin chân thành cảm ơn.

Nhóm thực hiện

**CHƯƠNG 1: TỔNG QUAN**

* 1. **Tổng quan về vấn đề được nghiên cứu**

Hiện nay, mạng Lan đã có những tiến bộ vượt bậc và ngày càng phổ biến hơn trong đồi sống sinh hoat. Điều này làm cho nhu cầu liên lạc và trao đổi thông tin thông qua mạng Lan ngày càng lớn hơn. Vì lẽ đó nên chương trình Chat trên mạng Lan được xây dựng để đáp ứng phần nào những nhu cầu cấp thiết đó. Chính vì thế, phần mềm Chat do nhóm em thực hiện hy vọng sẽ đáp ứng một phần nào.

* 1. **Nhiệm vụ đồ án**

Xây dựng chương trình Chat hoạt động trong mạng Lan với các chức năng cơ bản như: đăng ký, đăng nhập tài khoản để gửi tin nhắn, gửi tập tin, gửi tin nhắn cho nhiều người.

* Đối tượng nghiên cứu: Tìm hiểu được cơ chế hoạt động của Socket và Thread, từ đó viết ứng dụng Chat trong mạng Lan.
* Phạm vi nghiên cứu: Chương trình Chat được xây dựng với khả năng gửi các văn bản qua lại giữa các user thông qua sự điều khiển của một Server trong mạng Lan.
  1. **Cấu trúc đồ án**
* CHƯƠNG 1: TỔNG QUAN

1.1 Tổng quan về vấn đề được nghiên cứu:

1.2 Nhiệm vụ đồ án:

1.3 Cấu trúc đồ án:

* CHƯƠNG 2: CƠ SỞ LÝ THUYẾT

2.1 Khái niệm địa chỉ và cổng:

2.2 Sơ lược về lập trình đa luồng:

2.2.1 Khái niệm Luồng (Thread)

2.2.2 Đồng bộ hóa

2.2.3 Deadlock

* CHƯƠNG 3: PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG

3.1 Phân tích hệ thống:

3.1.1 Phân tích nhu cầu thực tiễn:

3.1.2 Yêu cầu đề ra:

3.2 Thiết kế hệ thống:

3.2.1 Sơ đồ Usecase tổng thể:

3.2.2 Sơ đồ hoạt động:

* CHƯƠNG 4: TỔNG KẾT VÀ KIẾN NGHỊ

4.1 Kết quả đạt được:

4.2 Hạn chế:

4.3 Hướng phát triển:

4.4 Tài liệu tham khảo:

**CHƯƠNG 2: CƠ SỞ LÝ THUYẾT**

* 1. **Khái niệm địa chỉ và cổng**
* **Nguyên lý:**
* Trong một máy có rất nhiều ứng dụng muốn trao đổi với các ứng dụng khác thông qua mạng (ví dụ có 2 ứng dụng trong máy A muốn trao đổi với 2 ứng dụng trên máy B).
* Mỗi máy tính chỉ có duy nhất một đường truyền dữ liệu (để gửi và nhận).
* **Vấn đề:** Rất có thể xảy ra "nhầm lẫn" khi dữ liệu từ máy A gửi đến máy B thì không biết là dữ liệu đó gửi cho ứng dụng nào trên máy B?
* **Giải quyết:** Mỗi ứng dụng trên máy B sẽ được gán một số hiệu (mà ta vẫn quen gọi là cổng: Port), số hiệu cổng này từ 1🡪65535. Khi ứng dụng trên máy A muốn gửi cho ứng dụng nào trên máy B thì chỉ việc điền thêm số hiệu cổng (vào trường RemotePort) vào gói tin cần gửi. Trên máy B, các ứng dụng chỉ việc kiểm tra giá trị cổng trên mỗi gói tin xem có trùng với số hiệu cổng của mình (đã được gán – chính là giá trị Localport) hay không? Nếu bằng thì xử lý, còn trái lại thì không làm gì (vì không phải là của mình). Khi cần trao đổi dữ liệu cho nhau thì hai ứng dụng cần phải biết thông tin tối thiểu là địa chỉ (Address) và số hiệu cổng (Port) của ứng dụng kia.
  1. **Sơ lược về lập trình đa luồng**
     1. ***Khái niệm luồng (Thread)***
* Thread là đơn vị nhỏ nhất của đoạn mã có thể thi hành được mà thực hiện một công việc (task) riêng biệt.
* Một ứng dụng có thể chia thành nhiều task, mỗi task có thể được gán cho một thread.
* Nhiều thread được thực hiện đồng thời gọi là multithreading.
* Với cơ chế multithreading, ứng dụng có thể thực thi nhiều task đồng thời. Mỗi thread sẽ có một chỉ số xác định quyền ưu tiên, thread nào có quyền ưu tiên cao sẽ được thực thi trước, và thấp thì thực thi sau.
* Việc thực thi và đánh chỉ số ưu tiên sẽ được thực hiện tự động bởi OS để tránh một thread nào đợi quá lâu mà không được vào xử lý.
* **Các trạng thái của Thread:**
  + Born: là trạng thái mà một thread mới tạo ra.
  + Ready: sau khi một thread được tạo ra, nó ở trong trạng thái sẵn sàng đợi phương thức **start()** được gọi.
  + Running: thread đi vào trạng thái này khi nó bắt đầu thực thi.
  + Sleeping: thread có thể tạm dừng tạm thời khi gọi phương thức **sleep()**. Nó trở nên sẵn sàng khi thời gian ngủ hết hạn.
  + Waiting: thread ở trạng thái này khi gọi phương thức **waiting()**. Được sử dụng khi nhiều thread chạy đồng thời.
  + Blocked: thread chuyển sang trạng thái bị chặn khi nó chờ đợi cho một sự kiện như I/O hoạt động.
  + Dead: thread ở trạng thái chết sau khi phương thức run() hoàn thành hoặc phương thức stop() được gọi.

***2.2.2 Đồng bộ hóa***

* Đôi khi 2 hoặc nhiều thread có thể truy cập vào một tài nguyên (bộ nhớ, file,...) cùng một lúc. Chẳng hạn, một thread x đọc dữ liệu từ một file, trong khi một thread y thay đổi dữ liệu trong cùng một file → có khả năng x đọc dữ liệu cũ (chưa được cập nhật). Trong trường hợp này, dữ liệu có thể trở nên không phù hợp (sai lạc).
* Để đảm bảo rằng một nguồn tài nguyên chia sẻ được sử dụng, bởi chỉ có một thread tại bất kỳ thời điểm nào, ta sử dụng đồng bộ hóa (synchronization). Đồng bộ hóa dựa trên khái niệm monitor (giám sát). Một monitor là một đối tượng được sử dụng như một ổ khóa loại trừ lẫn nhau. Chỉ có một thread đi vào monitor.
* Khi một thread đi vào monitor, điều này có nghĩa là thread đã thu được ổ khóa và tất cả các thread khác phải chờ cho đến khi thread này thoát khỏi monitor.
* **Có 2 cách đồng bộ hóa:**
* Phương thức đồng bộ (synchronized methods).
* Đồng bộ theo khối (synchronized blocks).
* **Đối với phương thức đồng bộ:** thêm **synchronized** để thực hiện đồng bộ hoá. Nếu bỏ modifier synchronized , tất cả các thread chạy đồng thời cùng phương thức trên cùng đối tượng. Gọi là điều kiện tranh chấp (race condition). Điều kiện tranh chấp xảy ra khi:
* Hai hoặc nhiều thread chia sẻ dữ liệu.
* Chúng đọc và ghi dữ liệu chia sẻ đồng thời.
* **Đồng bộ hóa trên toàn phương thức không phải là giải pháp chung cho các vấn đề đồng bộ. Vì đồng bộ trên toàn phương thức sẽ:**
* Làm mã được thực thi chậm hơn.
* Dễ gây deadlock.
* Quan trọng nhất, không phải nó luôn cần được bảo vệ khỏi sự thay đổi hoặc truy cập đồng thời 🡪 Đồng bộ theo khối.
* Đồng bộ theo khối đảm bảo rằng một phương thức có thể được gọi chỉ sau khi các thread hiện tại đã đi vào monitor thành công của đối tượng.
  + 1. ***Deadlock***
* Xảy ra khi 2 thread có phụ thuộc xoay vòng (phụ thuộc lẫn nhau) trên một cặp đối tượng đồng bộ hóa. Giả sử một luồng đang chạy theo đoạn mã sau, trong đó A, B là hai đối tượng tham chiếu mà cả hai luồng cần truy nhập.
* Có thể xảy ra biến cố sau: luồng đầu tiên yêu cầu một lock trên **A**, trong khi vào cùng thời điểm đó luồng thứ hai yêu cầu lock trên **B**. Một khoảng thời gian ngắn sau, luồng **A** gặp câu lệnh **lock(B)**, và ngay lập tức bước vào trạng thái ngủ, đợi cho lock trên **B** được giải phóng. Và tương tự sau đó, luồng thứ hai gặp câu lệnh **lock(A)** và cũng rơi vào trạng thái ngủ chờ cho đến khi lock trên A được giải phóng. Không may, lock trên **A** sẽ không bao giờ được giải phóng bởi vì luồng đầu tiên mà đã lock trên **A** đang ngủ và không thức dậy cho đến khi lock trên **B** được giải phóng điều này cũng không thể xảy ra cho đến khi nào luồng thứ hai thức dậy. Kết quả là deadlock. Cả hai luồng đều không làm gì cả, đợi lẫn nhau để giải phóng lock. Loại lỗi này làm toàn ứng dụng bị treo, ta phải dùng Task Manager để hủy nó.
* Deadlock có thể được tránh nếu cả hai luồng yêu cầu lock trên đối tượng theo cùng thứ tự. Trong ví dụ trên nếu luồng thứ hai yêu cầu lock cùng thứ tự với luồng đầu, **A** đầu tiên rồi tới b thì những luồng mà lock trên a đầu sẽ hoàn thành nhiệm vụ của nó sau đó các luồng khác sẽ bắt đầu.
* Kỹ thuật ngăn deadlock tốt nhất là tránh dùng kỹ thuật đồng bộ nếu không cần thiết.
* **Deadlock là một lỗi khó cho debug bởi hai nguyên nhân:**
  + Nó hiếm khi xảy ra.
  + Nó có thể liên quan đến nhiều hơn hai thread và hai đối tượng được đồng bộ. Nghĩa là deadlock có thể xảy ra thông qua một chuỗi các sự kiện phức tạp.

**CHƯƠNG 3: PHÂN TÍCH VÀ THIẾT KẾ YÊU CẦU HỆ THỐNG**

* 1. **Phân tích hệ thống**
     1. ***Phân tích nhu cầu thực tiễn***

Hiện nay, mạng LAN phát triển rất mạnh cả trong trường học, các cơ quan tổ chức và ở cả các hộ gia đình. Chính điều đó kéo theo nhu cầu liên lạc trao đổi thông tin trong mạng LAN cũng phát triển theo. Chính vì vậy, một chương trình Chat phục vụ cho nhu cầu liên lạc, trao đổi thông tin trong mạng LAN là rất cần thiết.

* + 1. ***Yêu cầu đề ra***

1. ***Đăng ký tài khoản:***

* Client gửi yêu cầu đăng ký tài khoản gồm username và password đến server.
* Server kiểm tra thông tin tài khoản đã được đăng ký hay chưa, nếu chưa thì lưu thông tin tài khoản vào cơ sở dữ liệu của mình (có thể sử dụng tập tin để lưu) và gởi thông báo đến client đăng ký thành công, ngược lại server gửi thông báo đến client tài khoản đã được đăng ký.

1. ***Chat (gởi dữ liệu text) với một tài khoản chỉ định (thông qua Username):***

* Client gửi thông tin tài khoản của clien muốn chat (username) đến server.
* Server kiểm tra tài khoản đã kết nối server chưa. Nếu chưa kết nối thì gửi thông báo đến client là tài khoản không online, ngược lại server làm trung gian thực hiện chuyển dữ liệu qua lại giữa 2 client này.
* Quá trình chat kết thúc nếu một trong 2 bên yêu cầu kết thúc chat.

1. ***Gửi tập tin bất kỳ:***

* Client gửi thông tin tài khoản của client muốn truyền tập tin (username) đến server.
* Server kiểm tra tài khoản đã kết nối server chưa. Nếu chưa kết nối thì gửi thông báo đến client là tài khoản không online, ngược lại thực hiện chuyển dữ liệu tập tin từ client gửi đến client nhận.
* Quá trình gửi tập tin kết thúc khi client nhận xong dữ liệu.

1. ***Chat nhóm:***

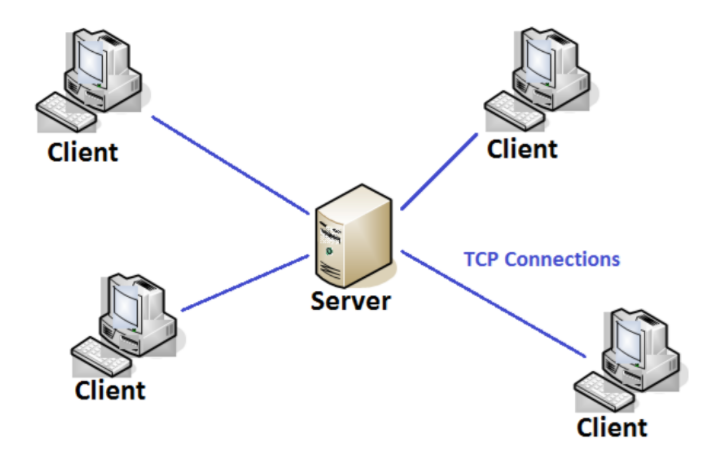
* Client gửi yêu cầu tạo nhóm chat (tên nhóm chat) đến server.
* Server chấp nhận yêu cầu và tạo nhóm với tên nhóm chat mà client đã gửi.
* Client gửi yêu cầu thêm một hoặc nhiều thành viên (username) vào nhóm chat.
* Server chấp nhận yêu cầu và thêm client vào nhóm chat.
* Client gửi dữ liệu chat nhóm đến server.
* Server thực hiện gửi dữ liệu này đến tất cả client trong nhóm.
* Client nếu muốn rời nhóm thì gửi yêu cầu rời nhóm đến server.
* Server tiếp nhận yêu cầu rời nhóm và xóa thành viên ra khỏi nhóm chat.
* Quá trình chat nhóm kết thúc khi tất cả thành viên rời khỏi nhóm.

1. ***Yêu cầu nâng cao:***

* Hỗ trợ giao diện java swing.
* Chức năng gửi tập tin cho phép hiển thị thông tin tập tin nếu file gởi là hình ảnh.
* Hỗ trợ âm thanh khi nhận được tin nhắn mới.
  + 1. ***Mục tiêu***

Sử dụng kỹ thuật lập trình socket và multithread trong Java xây dựng ứng dụng chat Client – Server đơn giản gồm các chức năng sau:

* Đăng ký User và quản lí tài khoản bằng database.
* Chat giữa Client và Client hoặc giữa nhiều Client với nhau thông qua Server để quản lý.
* Chat Group.
* Thông báo User online hoặc offline.
* Cho phép gửi nhận tệp có đuôi (.jpg).
* Phát triển thêm tính năng kết bạn (nếu có thời gian).
  1. **Thiết kế hệ thống**
     1. ***Mô hình Client – Server***

******

* ***TCP Socket:***
* Socket = IP + Port.

Trong đó: - IP: Địa chỉ IP của một máy tính trên mạng TCP/IP.

* + - Port: Một số nguyên hai byte không dấu.
* Máy truyền và máy nhận thiết lập một kết nối TCP trên cặp Socket.
* Sau khi hoàn tất quá trình thiết lập kết nối, hai máy tính có thể truyền/nhận dữ liệu với nhau.
* Hỗ trợ: Point – to – Point.
* Classes:
* ServerSocket: “lắng nghe” yêu cẩu kết nối.
* Socket: truyền/nhận dữ liệu.
  + 1. ***Sơ đồ Usecase tổng thể***

**Người quản trị Người dùng**

* **Danh sách các Stakeholder:**
  + Người dùng: Tất cả các người dùng đang vào máy khách, có chức năng kết nối tới máy chủ, gửi tin nhắn, gửi tập tin, đăng nhập, đăng ký.
  + Người quản trị: có thể truy cập và quản lý máy chủ, có quyền khởi động máy chủ.
    1. ***Sơ đồ hoạt động***
       1. ***Sơ đồ hoạt động kết nối tới máy chủ***



**Kết nối tới máy chủ**

**Kết nối thành công**

**Sai**

**Đúng**

**Mở chức năng đăng ký, đăng nhập**



* + - 1. ***Sơ đồ hoạt động đăng ký***



**Điền username, password và confirm password**

**Kiểm tra username, password, confirm password hợp lệ**

**Sai**

**Đúng**

**Gửi username, password đăng ký đến Server**

**Kiểm tra username có trùng hay không**

**Tồn tại**

**Không tồn tại**

**Thông báo đăng ký thành công và quay lại trang đăng nhập**



* + - 1. ***Sơ đồ hoạt động đăng nhập***



**Điền username và password**

**Gửi username và password đến Server**

**Kiểm tra username và password**

**Không tồn tại**

**Tồn tại**

**Mở ứng dụng chat**



* + - 1. ***Sơ đồ hoạt động gửi tin nhắn***



**Nhập tin nhắn và chọn đối tượng cần gửi (người dùng hoặc nhóm)**

**Nhấn nút gửi tin nhắn**

**Tin nhắn được gửi đến nơi cần gửi**



* + - 1. ***Sơ đồ hoạt động gửi tập tin***



**Nhấn vào nút chọn đường dẩn**

**Lấy đường dẫn tập tin**

**Xác thực kích thước tập tin**

**<<signal sending>>**

**Kích thước tập tin quá lớn**

**Hiển thị hộp thoại thông báo đến máy nhận tập tin**

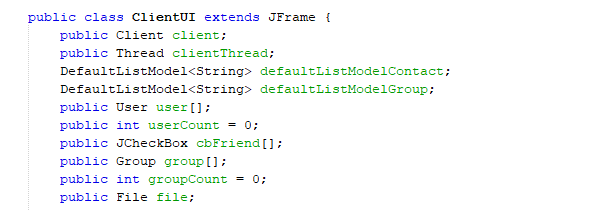
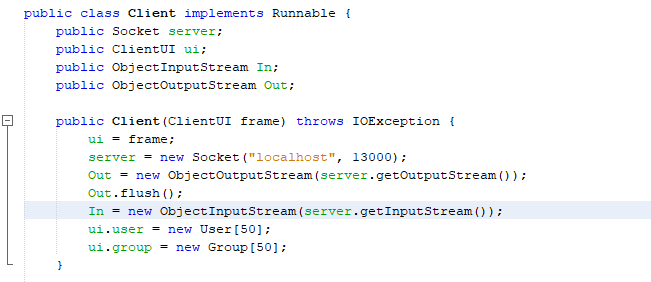
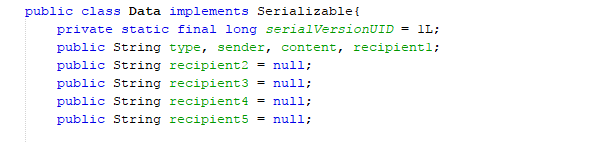
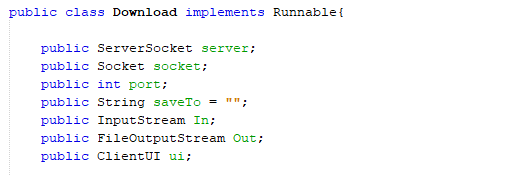
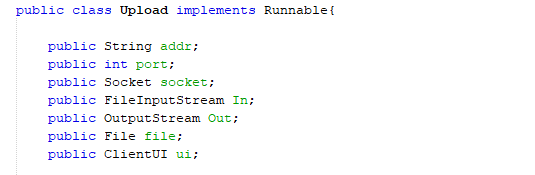
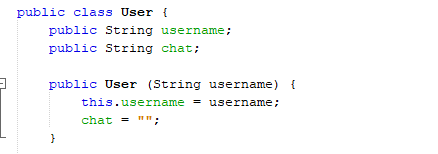
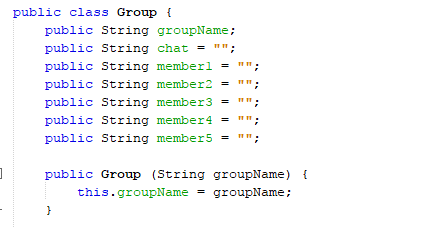
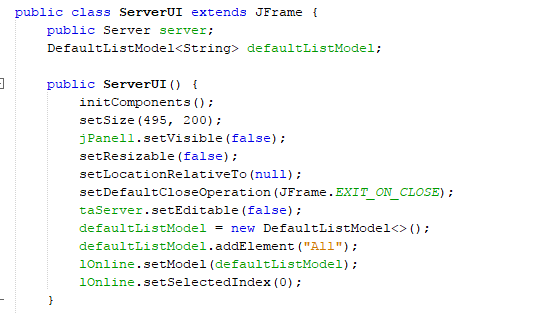
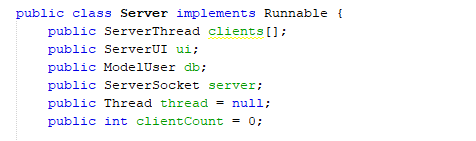
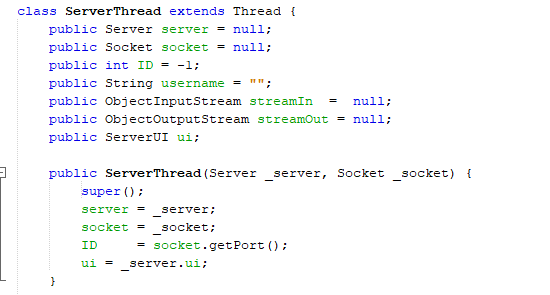
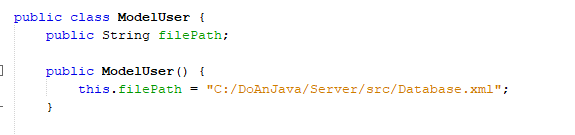
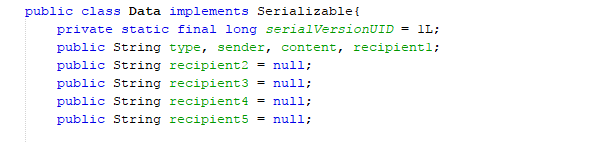
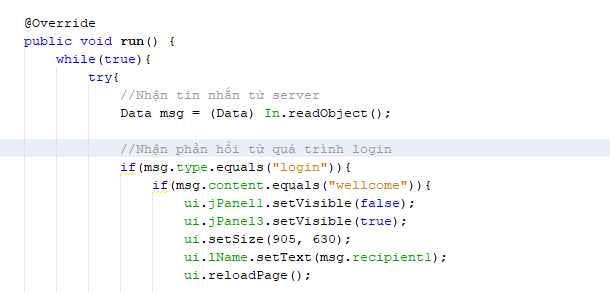
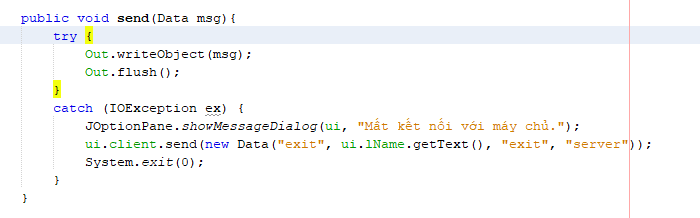
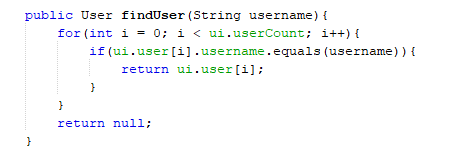
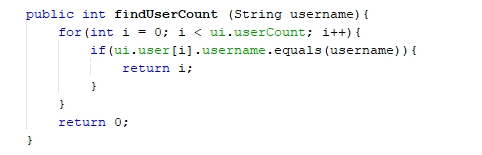
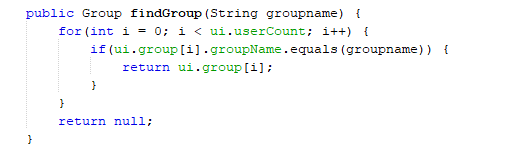
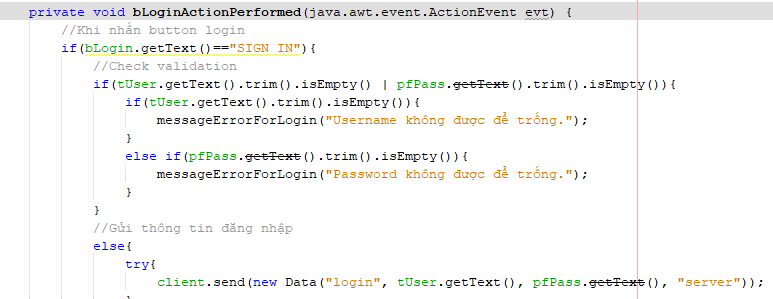
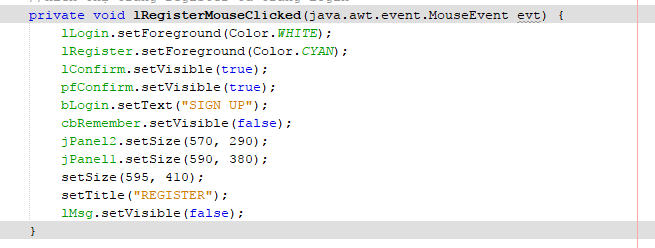
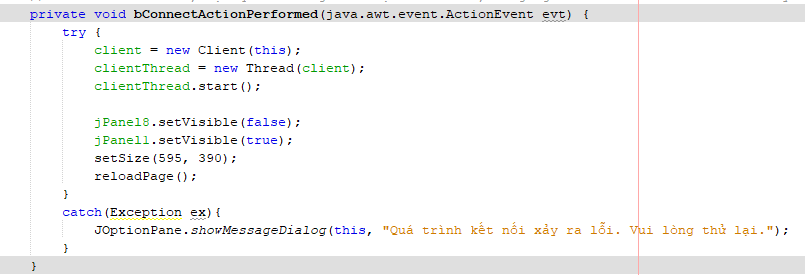
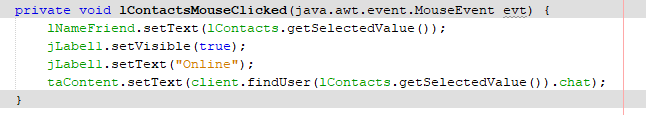
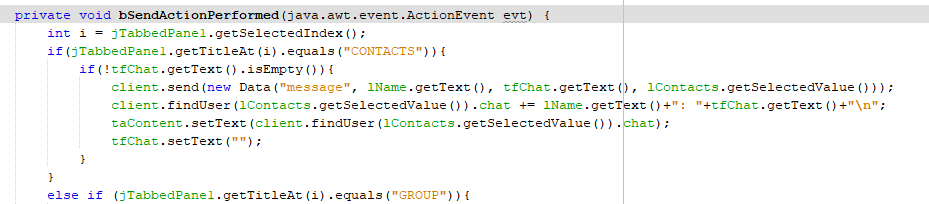
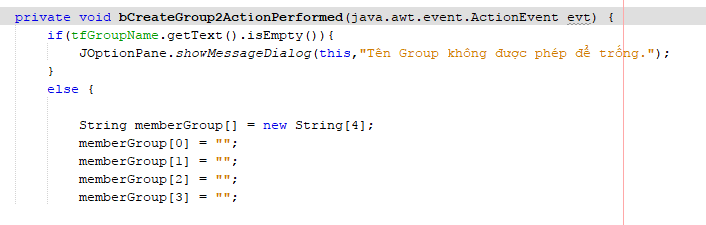
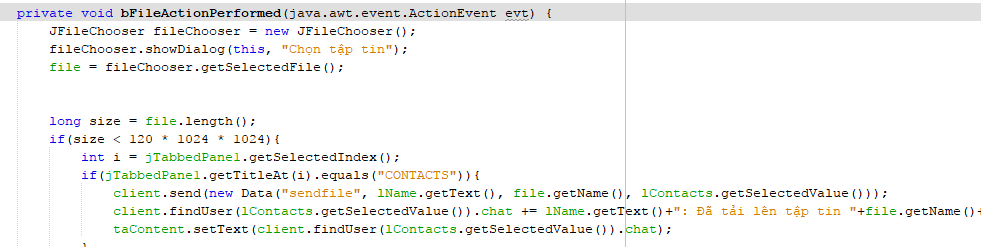
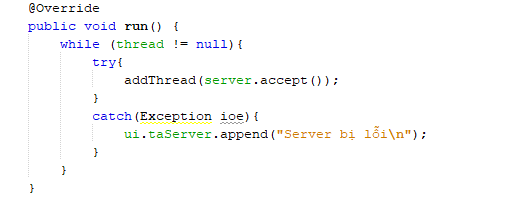
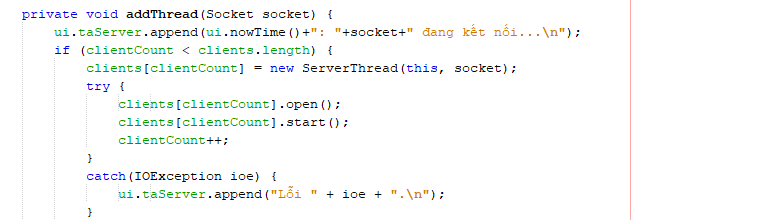
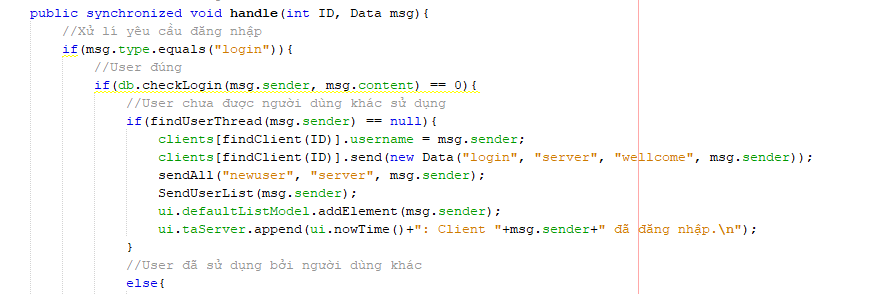
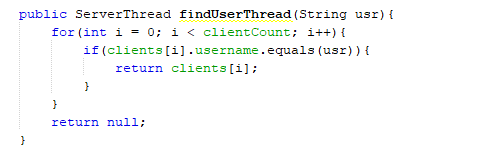
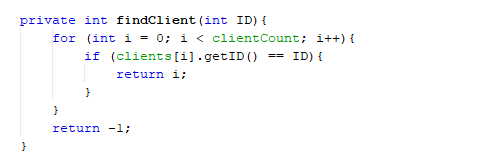
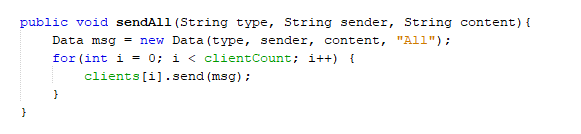
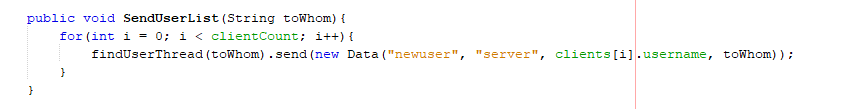
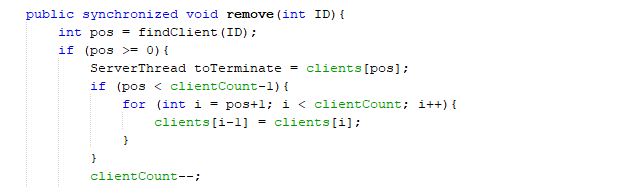
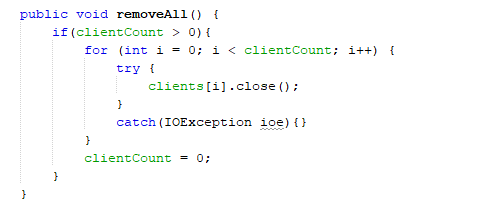
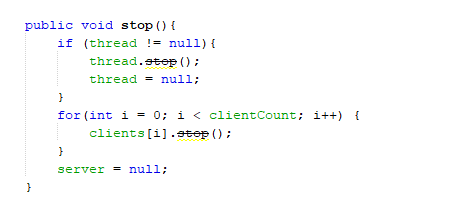
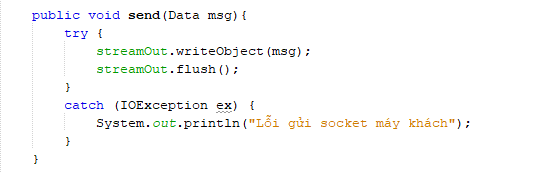
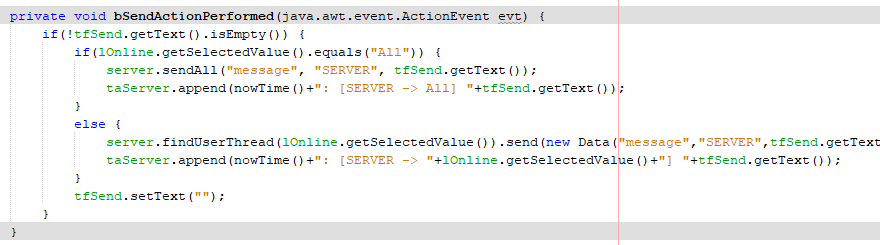
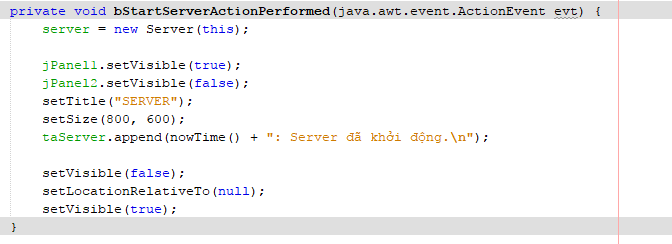
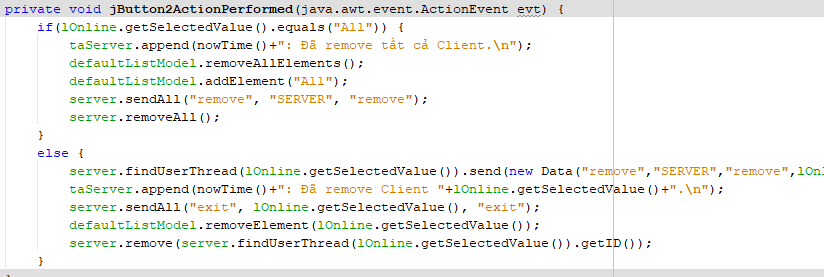
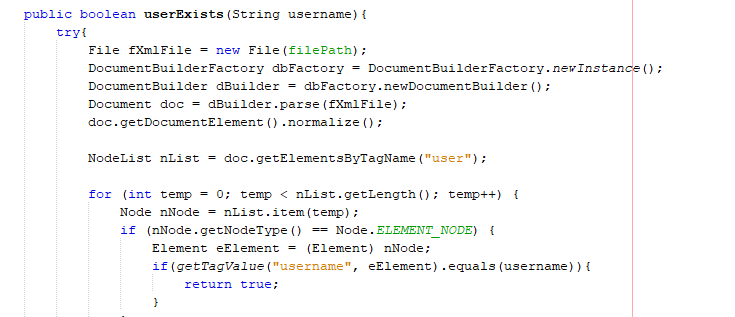
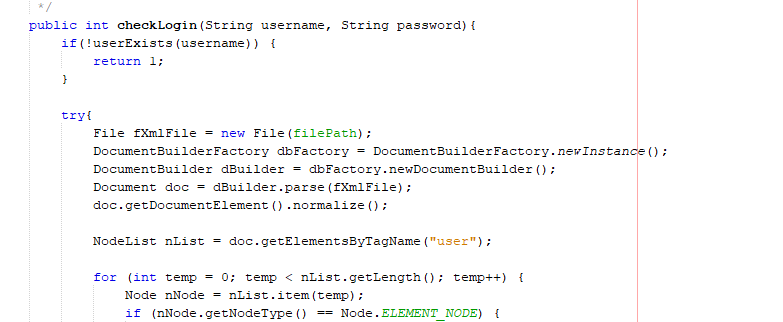
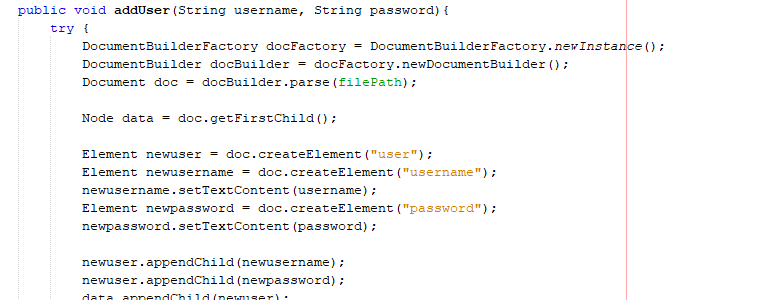
**Chấp nhận tập tin không?**

**<<signal sending>>**

**Tập tin yêu cầu bị từ chối**

**Chỉ ra đường dẫn được lưu xuống và tập tin được lưu xuống máy nhận**



* 1. **Mô tả các Class & Object sử dụng trong đồ án**
     1. ***Client***
* **Class ClienUI: tạo giao diện swing cho ứng dụng.**
* **Class Client: Kết nối với server, tạo quá trình truyền nhận dữ liệu và liên tục lắng nghe tín hiệu từ Server và xử lí các yêu cầu.**
* **Class Data: là đối tượng để lưu dữ liệu sau đó truyền đi từ Client sang Server.**
* **Class Download: tải file được gửi từ client xuống máy tính.**
* **Class Upload: tải file từ máy tính lên để gửi sang client khác.**
* **Class User: là đối tượng để lưu tên và các tin nhắn của một user.**
* **Class Group: là đối tượng để lưu danh sách thành viên và tin nhắn của group.**
  + 1. ***Server***
* **Class ServerUI: tạo giao diện swing cho Server.**
* **Class Server: Khởi tạo và chạy Server, sau đó liên tục lắng nghe các kết nối từ client, tạo quá trình gửi nhận dữ liệu cho client đó.**
* **Class ServerThread: liên tục lắng nghe dữ liệu được gửi đến từ client và xử lí chúng.**
* **Class ModelUser: tương tác với file database để ghi thông tin đăng ký, kiểm tra thông tin đăng nhập.**
* **Class Data: là đối tượng để lưu dữ liệu sau đó truyền đi từ Server sang Client.**
  1. **Các hàm xử lý và ý nghĩa của hàm**
     1. ***Client***
* **Hàm run trong class client: lắng nghe dữ liệu được gửi đến từ Server và xử lý chúng.**
* **Hàm send (tham số là đối tượng Data cần gửi): gửi đối tượng Data đến Server.**
* **Hàm findUser (tham số là tên user cần tìm): Tìm client thông qua tên user.**
* **Hàm findUserCount (tham số là tên user cần tìm): tìm vị trí của user đó trong list user thông qua tên user và trả về vị trí của user đó.**
* **Hàm findGroup (tham số là tên group cần tìm): tìm group thông qua tên group.**
* **Hàm bLoginActionPerformed: bắt sự kiện click button đăng ký hay button đăng nhập, sau đó check dữ liệu đăng nhập hoặc đang ký, nếu hợp lệ thì gửi đến Server.**
* **Hàm lRegisterMouseClicked: bắt sự kiện click vào jlabel đăng ký, và hiện giao diện đăng ký.**
* **Hàm bConnectActionPerformed: bắt sự kiện click button connect, thực hiện kết nối với Server.**
* **Hàm lContactsMouseClicked: bắt sự kiện click vào danh sách tên user hoặc group để hiện đoạn chat tương ứng ra màn hình.**
* **Hàm bSendActionPerformed: bắt sự kiện click button gửi tin nhắn, thực hiện gửi tin nhắn và người hoặc group cần gửi đến server.**
* **Hàm bCreateGroup2ActionPerformed: bắt sự kiện click button tạo group, và gửi thông tin tên group và thành viên group đến server.**
* **Hàm bFileActionPerformed: bắt sự kiện click button gửi file, kiểm tra kích thước file, gửi file đến server.**
  + 1. ***Server***
* **Hàm run trong class Server: lắng nghe kết nối từ các client, và thêm vào danh sách client.**
* **Hàm addThread: thêm các client muốn kết nối đến vào danh sách client, tạo quá trình gửi và nhận dữ liệu với clien đó và chạy nó.**
* **Hàm handle (tham số là id của client và dữ liệu mà client đó gửi đến): xử lí các dữ liệu từ client gửi đến.**
* **Hàm findUserThread (tham số là tên user cần tìm): tìm user thông qua tên user.**
* **Hàm findClient (tham số là id của user cần tìm): tìm vị trí của user trong mảng danh sách các user thông qua id.**
* **Hàm sendAll (tham số là các message muốn gửi đi): gửi dữ liệu đến tất cả các client.**
* **Hàm SendUserList (tên user vừa đăng nhập): thông báo đến tất cả các client khi có client mới đăng nhập.**
* **Hàm remove (truyền vào id user): xóa user ra khỏi danh sách user.**
* **Hàm removeAll: xóa toàn bộ danh sách user.**
* **Hàm stop: dừng một user.**
* **Hàm send (tham số là đối tượng Data): gửi đối tượng Data đến client.**
* **Hàm bSendActionPerformed: bắt sự kiện click button send, gửi tin nhắn đến client.**
* **Hàm bStartServerActionPerformed: bắt sự kiện kiện click button khởi động server.**
* **Hàm jButton2ActionPerformed: bắt sự kiện click button remove user, và remove user được chọn khỏi danh sách user.**
* **Hàm userExists: kiểm tra username đăng ký có tồn tại không.**
* **Hàm checkLogin: kiểm tra username, password nhập vào có đúng không.**
* **Hàm addUser: thêm user đăng ký vào database.**
  1. **Sơ đồ gọi hàm khi thực hiện chức năng**
     1. ***Đăng ký***

**Server**

**Client**

**bLoginActionPerformed**

**run (trong class ServerThread)**

**handle**

**send**

**addUser**

**userExists**

**findClient**

**run (trong class Client)**

**send**

* + 1. ***Đăng nhập***

**Server**

**Client**

**bLoginActionPerformed**

**run (trong class ServerThread)**

**handle**

**send**

**checkLogin**

**findUserThread**

**findClient**

**sendUserList**

**sendAll**

**run (trong class Client)**

* + 1. ***Gửi tin nhắn đến Client***

**Client gửi**

**Server**

**bSendActionPerformed**

**run (trong class ServerThread)**

**handle**

**send**

**findUserThread**

**Client nhận**

**run (trong class Client)**

**send**

**findUser**

* + 1. ***Gửi tin nhắn đến Group***

**Client gửi**

**Server**

**run (trong class ServerThread)**

**bSendActionPerformed**

**handle**

**findUserThread**

**Client nhận**

**send**

**run (trong class Client)**

**findGroup**

* + 1. ***Tạo Group***

**Client**

**Server**

**bCreateGroup2ActionPerformed**

**findUserThread**

**send**

**send**

**run (trong class Client)**

**Group**

* + 1. ***Gửi file***

**Client nhận**

**Server**

**Client gửi**

**bFileActionPerformed**

**send**

**handle**

**run (trong class ServerThread)**

**send**

**Download**

**findUserThread**

**findUser**

**run (trong class Client)**

**send**

**Upload**

**CHƯƠNG 4: TỔNG KẾT VÀ KIẾN NGHỊ**

**4.1 Kết quả đạt được**

* Đã tìm hiểu được cách thức lập trình Socket và lập trình đa luồng.
* Đã tìm hiểu được cách thức hoạt động của một chương trình Chat đơn giản.
* Đáp ứng được các yêu cầu chức năng đơn giản.
* Giao diện rất dễ nhìn, thân thiện cho người sử dụng.
* Mặc dù phần mềm đã hoàn tất, nhưng vẫn không tránh khỏi sai sót, rất mong sự góp ý của thầy để đề tài đồ án của nhóm em hoàn thiện hơn nữa và gần với thực tế hơn.

**4.2 Hạn chế**

* Chưa hiển thị được hình ảnh khi gửi file .jpg.

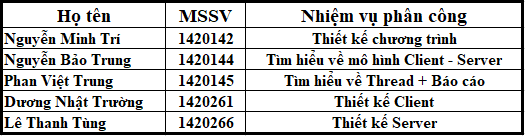
**4.3 Hướng phát triển**

* Tiếp tục nghiên cứu để phát triển phần mềm hoàn thiện hơn, sửa chữa các sai xót.
* Tiếp tục nghiên cứu để làm cho phần mềm thêm nhiều tính năng mới. Cụ thể là:
* Hỗ trợ chức năng Voice Chat và Webcam.
* Hỗ trợ thêm biểu tượng emotion (biểu tượng cảm xúc).
* Hỗ trợ thêm tính năng đọc tin tức, bài viết trong phần mềm.

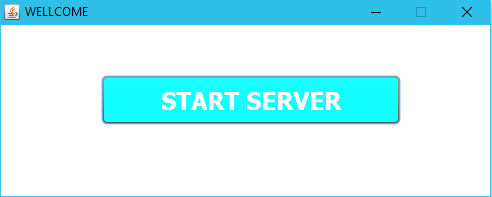
**4.4 Tài liệu tham khảo**

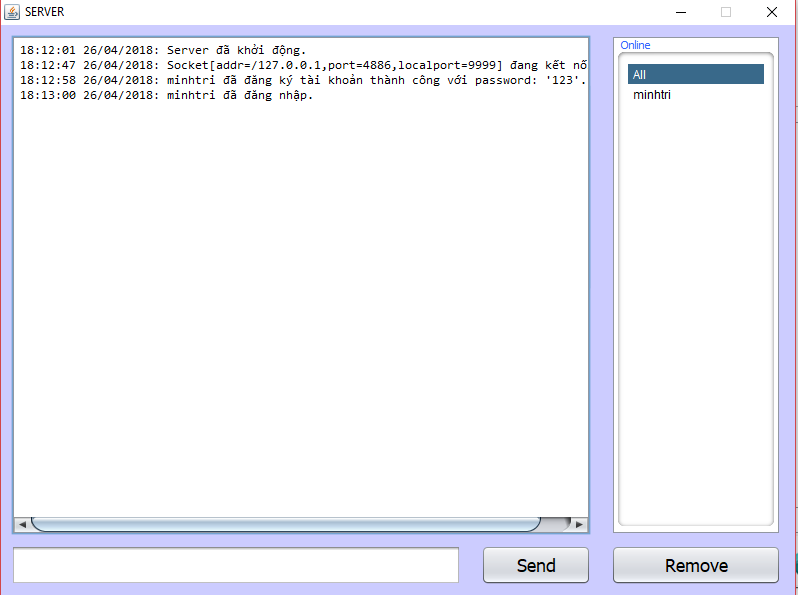
* Slide bài giảng:
* Bài giảng 3: Đối tượng – lớp trong Java.
* Bài giảng 6: Lập trình đa luồng.
* Bài giảng 7: Lập trình Socket.
* Các web như:
* <http://www.codeproject.com/>.
* <http://diendan.congdongcviet.com/>.
* <http://www.java2s.com/>
* <https://123doc.org//document/2551212-do-an-chat-qua-mang-bang-java.htm>.

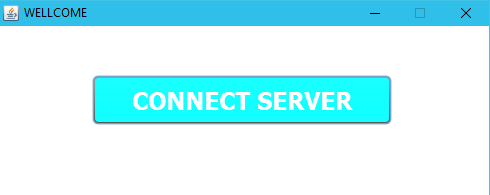
**PHÂN CÔNG CÔNG VIỆC**

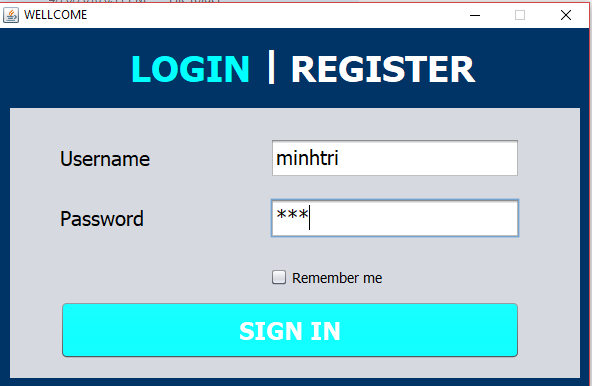


**PHỤ LỤC – GIAO DIỆN MÀN HÌNH PHẦN MỀM DEMO CHAT**

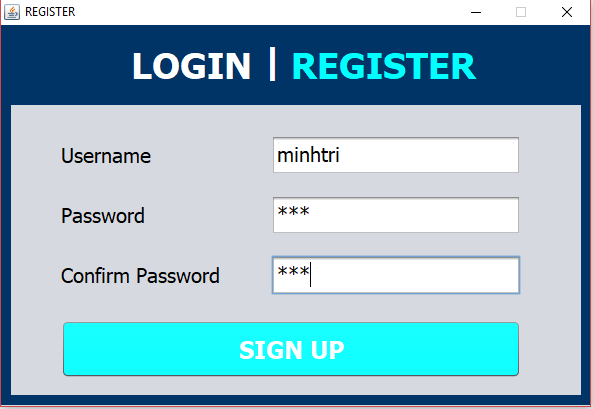
* + - 1. **Giao diện mở máy chủ:**
      2. **Máy chủ được thiết lập:**

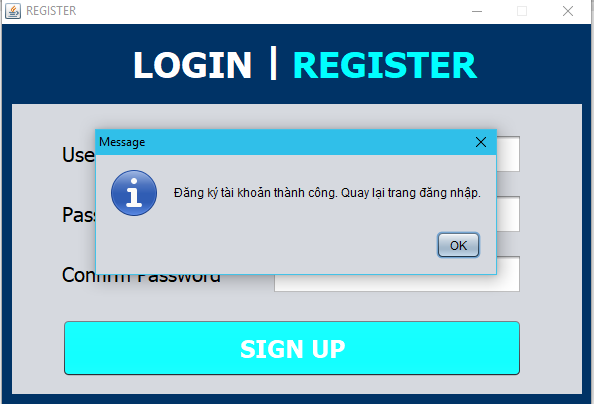


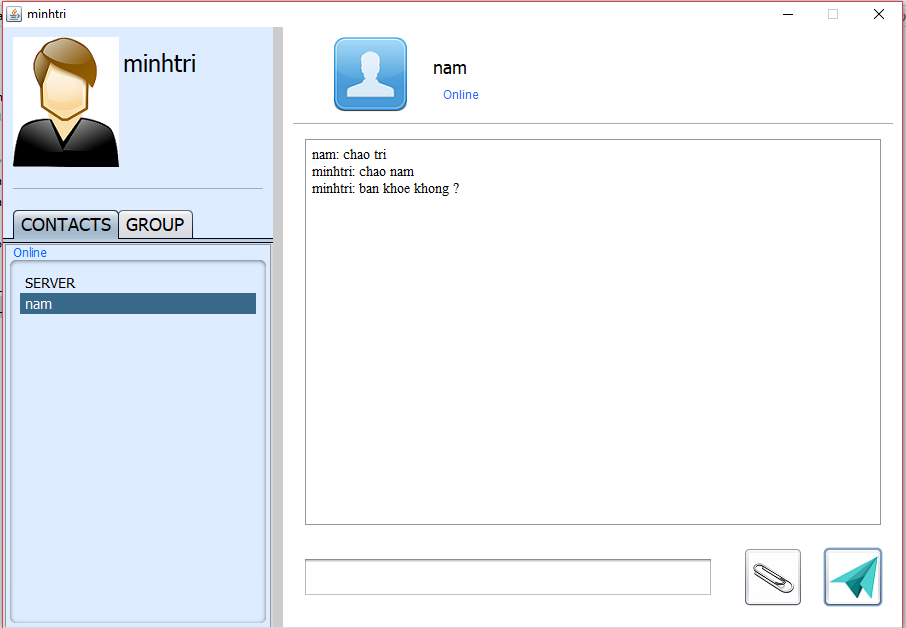
* + - 1. **Giao diện kết nối tới máy chủ:**
      2. **Giao diện đăng nhập:**



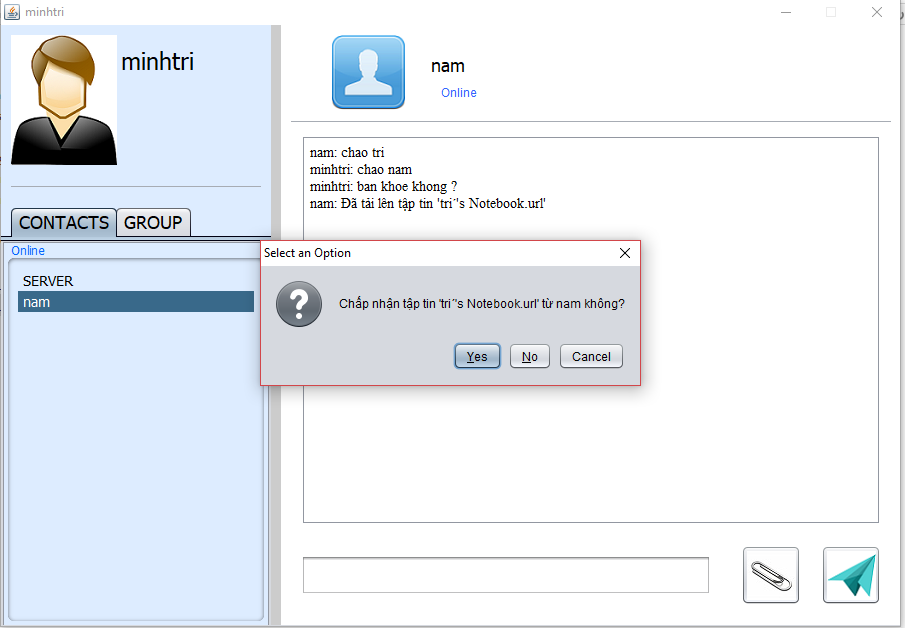
* + - 1. **Giao diện đăng ký:**



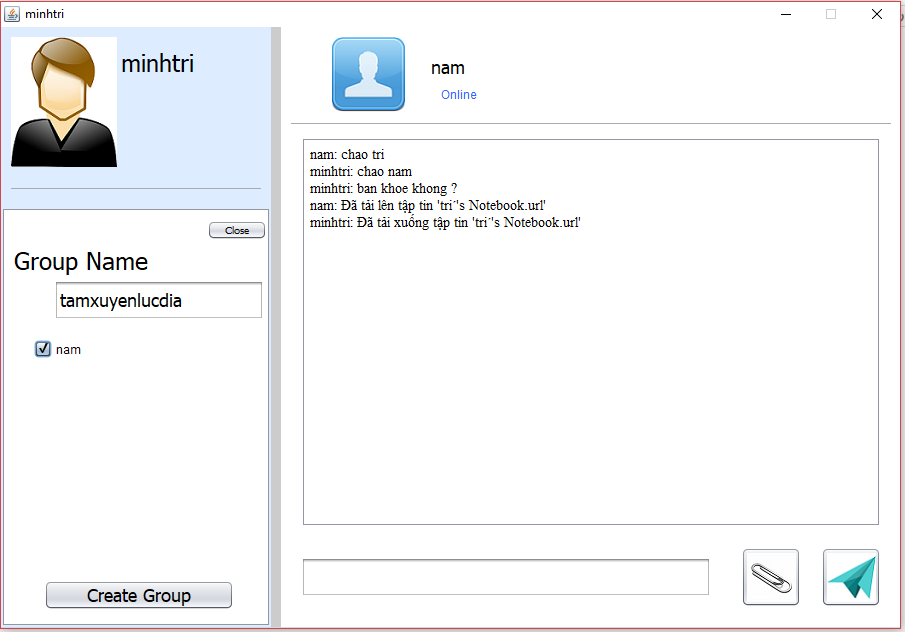
* + - 1. **Đăng ký thành công:**
      2. **Chat giữa hai Client:**



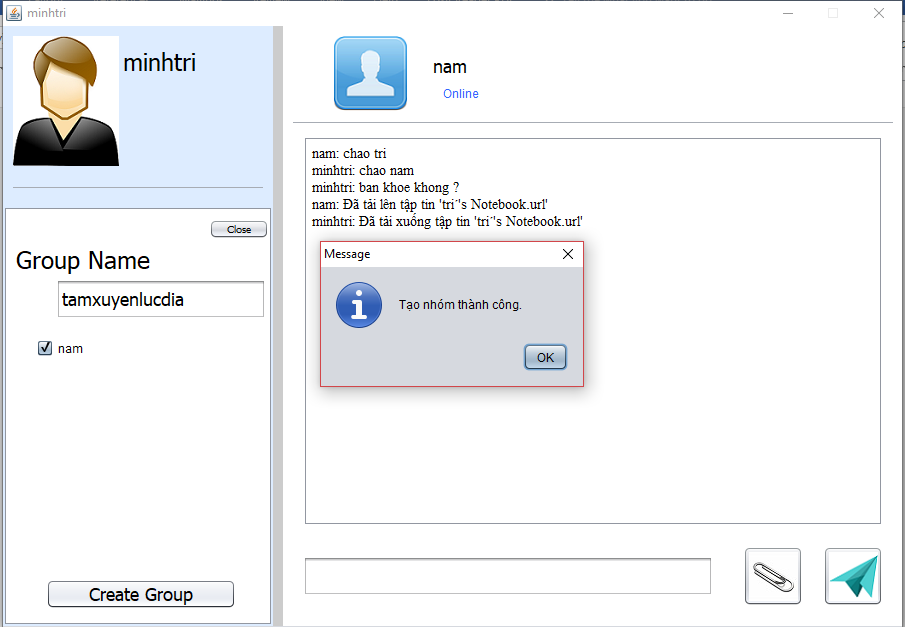
* + - 1. **Chức năng gửi tập tin:**



* + - 1. **Tạo Group:**



* + - 1. **Tạo Group thành công:**



* + - 1. **Server quản lý quá trình gửi nhận dữ liệu giữa các Client:**

