

**TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM
TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG
KHOA CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO
MÔN LẬP TRÌNH MẠNG CĂN BẢN**

**XÂY DỰNG CHƯƠNG TRÌNH CHAT
BẰNG GIAO THỨC TCP GIỮA HAI MÁY
TÍNH VÀ CÁC ỨNG DỤNG GIỮA
CLIENT VÀ SERVER**

Người hướng dẫn: **THẦY BÙI QUY ANH**

Người thực hiện: **CAO NGUYỄN KỲ DUYÊN – 51900491**

ĐỖ THỊ HOÀI THU – 51900563

NGUYỄN NGỌC THỦY VY – 51900579

Lớp : 19050402

19050401

Khoá : 23

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2021

**TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM
TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG
KHOA CÔNG NGHỆ THÔNG TIN**



BÁO CÁO
MÔN LẬP TRÌNH MẠNG CĂN BẢN
XÂY DỰNG CHƯƠNG TRÌNH CHAT
BẰNG GIAO THỨC TCP GIỮA HAI MÁY
TÍNH VÀ CÁC ỨNG DỤNG GIỮA
CLIENT VÀ SERVER

Người hướng dẫn: **THẦY BÙI QUY ANH**

Người thực hiện: **CAO NGUYỄN KỲ DUYÊN – 51900491**

ĐỖ THỊ HOÀI THU – 51900563

NGUYỄN NGỌC THỦY VY – 51900579

Lớp : 19050402

19050401

Khoá : 23

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2021

LỜI CẢM ƠN

Nhóm em xin chân thành gửi lời cảm ơn đến thầy Bùi Quy Anh đã giúp đỡ, hướng dẫn và dìu dắt chúng em trong quá trình học tập và tìm hiểu về bộ môn “Lập trình mạng căn bản”. Nhờ như vậy, chúng em có thể thực hiện bài báo cáo này một cách tốt nhất và có thể đạt được một kết quả tốt nhất. Một lần nữa, nhóm em xin bày tỏ lòng biết ơn của mình đến với thầy Bùi Quy Anh và chúc thầy sẽ luôn thành công trên con đường dạy học của mình.

Bài làm của chúng em là sản phẩm của những sinh viên ít kinh nghiệm nên chắc hẳn còn nhiều thiếu sót. Chúng em rất chân thành đón nhận và cảm ơn tất cả các ý kiến đóng góp của thầy. Nhờ những ý kiến đóng góp của thầy mà các bài làm sau của chúng em sẽ đạt được sự hoàn thiện cao hơn.

Chúng em xin chân thành cảm ơn!

ĐỒ ÁN ĐƯỢC HOÀN THÀNH TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG

Tôi xin cam đoan đây là sản phẩm đồ án của riêng tôi và được sự hướng dẫn của Thầy Bùi Quy Anh. Các nội dung nghiên cứu, kết quả trong đề tài này là trung thực và chưa công bố dưới bất kỳ hình thức nào trước đây. Những số liệu trong các bảng biểu phục vụ cho việc phân tích, nhận xét, đánh giá được chính tác giả thu thập từ các nguồn khác nhau có ghi rõ trong phần tài liệu tham khảo.

Ngoài ra, trong đồ án còn sử dụng một số nhận xét, đánh giá cũng như số liệu của các tác giả khác, cơ quan tổ chức khác đều có trích dẫn và chú thích nguồn gốc.

Nếu phát hiện có bất kỳ sự gian lận nào tôi xin hoàn toàn chịu trách nhiệm về nội dung đồ án của mình. Trường đại học Tôn Đức Thắng không liên quan đến những vi phạm tác quyền, bản quyền do tôi gây ra trong quá trình thực hiện (nếu có).

TP. Hồ Chí Minh, ngày tháng năm

Tác giả

(ký tên và ghi rõ họ tên)

Cao Nguyễn Kỳ Duyên

Đỗ Thị Hoài Thu

Nguyễn Ngọc Thủy Vy

PHẦN XÁC NHẬN VÀ ĐÁNH GIÁ CỦA GIẢNG VIÊN

Phần xác nhận của GV hướng dẫn

Tp. Hồ Chí Minh, ngày tháng năm
(kí và ghi họ tên)

Phần đánh giá của GV chấm bài

Tp. Hồ Chí Minh, ngày tháng năm
(kí và ghi họ tên)

TÓM TẮT

Chúng ta đã chứng kiến nhiều thay đổi to lớn trong ngành công nghệ thông tin và những thay đổi này đã góp phần tác động sâu sắc vào cuộc sống và cách sống của chúng ta. Máy tính đã trở nên vô cùng quan trọng trong các công việc hàng ngày như học tập, giao dịch, gửi thư,... Với nhu cầu trao đổi thông tin trong mạng toàn cầu Internet, Socket đã được tạo ra để xây dựng các ứng dụng theo kiến trúc Client-Server. Dữ liệu được truyền qua Internet bằng các gói có kích thước giới hạn được gọi là datagrams. Mỗi dữ liệu đều chứa một tiêu đề và một tải trọng. Phần đầu đề bao gồm địa chỉ và cổng mà gói dữ liệu đang đi tới, địa chỉ và cổng từ đó nó đến, và nhiều thông tin quản lý nội bộ khác được sử dụng để đảm bảo truyền tải đáng tin cậy. Tuy nhiên, do các biểu đồ dữ liệu có chiều dài giới hạn, nên thường cần chia dữ liệu thành nhiều gói dữ liệu và lắp ráp lại tại đích. Theo dõi việc này để phân chia dữ liệu thành các gói, tạo ra các tiêu đề, phân tích đầu của các gói nhận, theo dõi các gói có và chưa nhận được, và v.v... là rất nhiều công việc và đòi hỏi nhiều mã phức tạp. Sockets cho phép người lập trình xử lý kết nối mạng như một dòng khác mà trên đó có thể viết được byte và từ đó có thể đọc được byte. Sockets bảo vệ người lập trình khỏi các chi tiết cấp thấp của mạng, chẳng hạn như phát hiện lỗi, kích cỡ gói, truyền tải lại gói, địa chỉ mạng, v.v.

Do đó, mục đích của bài báo cáo này là nghiên cứu, phân tích những đặc điểm của các Socket từ đó nêu ra các cơ sở lý thuyết ở chương 1. Trên cơ sở đó, đề xuất xây dựng một mô hình ứng dụng chat bằng TCP giữa các ứng dụng client và server ở chương 2. Ở mô hình demo, chúng em sẽ tiến hành xây dựng một ứng dụng chat cùng với một số tính năng cơ bản.

MỤC LỤC

LỜI CẢM ƠN	1
PHẦN XÁC NHẬN VÀ ĐÁNH GIÁ CỦA GIẢNG VIÊN	3
TÓM TẮT	4
MỤC LỤC	5
DANH MỤC CÁC BẢNG BIỂU, HÌNH VẼ, ĐỒ THỊ	7
CHƯƠNG 1 – CƠ SỞ LÝ THUYẾT	9
1.1 Socket for client	9
1.1.1 Tổng quan về Socket.....	9
1.1.2 The Socket Class.....	9
a. Xây dựng phương thức	10
b. Nhập thông tin vào Socket.....	11
c. Closing the Socket	13
d. Tùy chọn Socket	14
1.1.3 Socket Exceptions.....	15
1.1.4 Socket Addresses	16
1.2 Socket for server	18
1.2.1 The Constructors.....	18
1.2.2 Accepting and Closing Connections.....	20
1.2.3 The get Methods	22
1.2.4 Socket Options	23
1.2.5 The Object Methods.....	25
1.2.6 Implementation	26
CHƯƠNG 2 _ MÔ HÌNH DEMO	29
CHƯƠNG 3 _ KẾT LUẬN.....	56

DANH MỤC KÍ HIỆU VÀ CHỮ VIẾT TẮT

CÁC KÝ HIỆU

CÁC CHỮ VIẾT TẮT

TCP : Transmission Control Protocol

DANH MỤC CÁC BẢNG BIỂU, HÌNH VẼ, ĐỒ THỊ

DANH MỤC HÌNH VÀ BẢNG

Bảng 1.2.4 : Socket Option	25
Bảng 1.2.5 : Object Method	26
Hình 2.1 Mở XAMPP và nhấn “start” Apache và MySQL.	29
Hình 2.2 : Tạo cơ sở dữ liệu tên “ltm_final”	29
Hình 2.3 : Import file ltm_final.sql vào cơ sở dữ liệu ltm_final.	30
Hình 2.4 : Khởi tạo hai bảng “accounts” và “users” thành công.	30
Hình 2.5 Chạy file Server.java.	31
Hình 2.6 Giao diện file Server.java.	31
Hình 2.7 Chạy file Main.java.	31
Hình 2.8 : Giao diện đăng nhập của file Main.java.	32
Hình 2.9 : Giao diện đăng ký tài khoản của file Main.java.	33
Hình 2.10 : Đăng ký tài khoản “a” với mật khẩu là “password”.	33
Hình 2.11 : Giao diện Chat khi đăng ký tài khoản thành công.	34
Hình 2.12 : Server sẽ hiển thị các thông báo khi có các máy khách kết nối với server và các thông tin đăng ký tài khoản.	35
Hình 2.13 : Hệ thống Chat sẽ hiển thị thông báo tên đăng nhập đã tồn tại để người dùng chọn tên đăng nhập khác để đăng ký tài khoản.	35
Hình 2.15 : Hệ thống chat sẽ hiển thị các tài khoản đã đăng ký trước đó ở giao diện Chat của người dùng “b”.	37
Hình 2.16: Khi này, giao diện của người dùng “a” cũng sẽ hiển thị tài khoản người dùng “b”.	38
Hình 2.17 : Người dùng “a” thực hiện thao tác gửi tin nhắn cho người dùng “b”. Ngoài nội dung tin nhắn thì hệ thống Chat còn hiển thị thêm thời gian gửi tin nhắn đó.	38
Hình 2.18 : Phía người dùng “b” nhận được tin nhắn từ người dùng “a”	39

Hình 2.19 : Đoạn tin nhắn giữa người dùng “a” và người dùng “b”.	40
Hình 2.20 : Giao diện hệ thống Chat khi gõ một tin nhắn dài.	41
Hình 2.21 : Hệ thống Chat có thanh cuộn để người dùng có thể kéo lên và kéo xuống, dễ dàng cho việc đọc tin nhắn.	42
Hình 2.22 :Sau đó đăng ký thêm các tài khoản người dùng hệ thống Chat, và những thông tin này đều được lưu trên cơ sở dữ liệu ltm_final.....	42
Hình 2.23 Các thư viện sử dụng trong hệ thống Chat.....	55

CHƯƠNG 1 – CƠ SỞ LÝ THUYẾT

1.1 Socket for client

1.1.1 Tổng quan về Socket

Trong Java, các lớp URL và URLConnections cung cấp một cơ chế nâng cao để truy cập tài nguyên trên Internet. Tuy nhiên, đôi khi các chương trình của bạn cần giao tiếp với nhau qua mạng cục bộ, chẳng hạn khi bạn muốn viết một ứng dụng máy khách-máy chủ. TCP (Transmission Control Protocol) cung cấp một kênh giao tiếp đáng tin cậy mà các ứng dụng máy khách-máy chủ sử dụng để giao tiếp với nhau. Để có thể giao tiếp với nhau qua TCP, máy khách và máy chủ được thiết lập để kết nối với nhau thông qua Socket. Vậy Socket là gì?

Socket là điểm cuối của liên kết giao tiếp hai chiều giữa hai chương trình đang chạy trên mạng. Lớp Socket được sử dụng để biểu diễn kết nối giữa máy khách và máy chủ. Nó có thể thực hiện bảy hoạt động cơ bản sau:

- Kết nối với một máy từ xa
- Gửi dữ liệu
- Nhận dữ liệu
- Đóng một môi liên hệ
- Liên kết với một cổng
- Nghe xem có nhận được dữ liệu không
- Chấp nhận kết nối từ các máy từ xa trên cổng giới hạn

Client Sockets

Khi kết nối được thiết lập, các chủ thể địa phương và từ xa lấy các luồng đầu vào và đầu ra từ socket và sử dụng các luồng đó để gửi dữ liệu cho nhau.

Khi việc truyền dữ liệu hoàn tất, một hoặc cả hai bên sẽ đóng kết nối.

1.1.2 The Socket Class

a. Xây dựng phương thức

1. Một máy khách khởi tạo đối tượng Socket, xác định tên máy chủ (IP hoặc domain) và số cổng để kết nối.

```
Public Socket(String host, int port) throws UnknownHostException, IOException
// Trình xây dựng này tạo ổ cắm TCP đến cổng được chỉ định trên máy chủ được chỉ định và
cố gắng kết nối với máy chủ từ xa.
```

Ví dụ :

```
Socket socket = new Socket("HostName", PortNumber);
// gửi và nhận dữ liệu ...
```

2. Trình xây dựng tiếp theo giống hệt với hàm tạo trước đó, ngoại trừ việc máy chủ được biểu hiện bởi một đối tượng InetAddress

```
public Socket(InetAddress host, int port) throws IOException
// Xây dựng này tạo ra một ổ đĩa TCP cho một cổng định sẵn trên một chủ thể nhất định và
cố gắng kết nối
```

Ví dụ :

```
Java ▼
Socket socket = new Socket(InetAddress , PortNumber);
// gửi và nhận dữ liệu ...
```

3. Trình này sẽ kết nối đến máy chủ và cổng được chỉ định, tạo một socket trên máy chủ cục bộ tại địa chỉ và cổng được chỉ định.

```

    public Socket(String host, int port, InetAddress interface, int localPort) throws IOException, UnknownHostException
    try {
        InetAddress inward = InetAddress.getByName("router");
        Socket socket = new Socket("mail", 25, inward, 0);
        //Làm việc với sockets...
    }
    catch (UnknownHostException ex) {

        System.err.println(ex);
    }
    catch (IOException ex) {
        System.err.println(ex);
    }

```

4. Trình này giống hệt với trình trước đó, ngoại trừ máy chủ được biểu hiện bởi một đối tượng `InetAddress` thay vì một `String`.

```

Java
public Socket(InetAddress host, int port, InetAddress interface, int localPort) throws IOException
try {
    InetAddress inward = InetAddress.getByName("router");
    InetAddress mail = InetAddress.getByName("mail");
    Socket socket = new Socket(mail, 25, inward, 0);
    //Làm việc với sockets...
}
catch (UnknownHostException ex) {
    System.err.println(ex);
}
catch (IOException ex) {
    System.err.println(ex); }

```

b. Nhập thông tin vào Socket

public InetAddress getInetAddress()

Phương thức `getInetAddress()` này trả về địa chỉ của máy tính khác mà socket này được kết nối..

▪ Ví dụ :

```
Socket socket = new Socket("HostName", PortNumber)
InetAddress host = socket.getInetAddress( );
```

public int getPort()

Cho ta biết Socket đã được kết nối hoặc sẽ được vào cổng nào trên máy chủ từ xa

▪ Ví dụ :

```
Java ▾
Socket socket = new Socket("HostName", PortNumber);
int port = socket.getPort( );
```

public int getLocalPort()

Trả về cổng mà socket bị ràng buộc trên máy local.

▪ Ví dụ :

```
Socket socket = new Socket("HostName", PortNumber, true);
int localPort = socket.getLocalPort( );
```

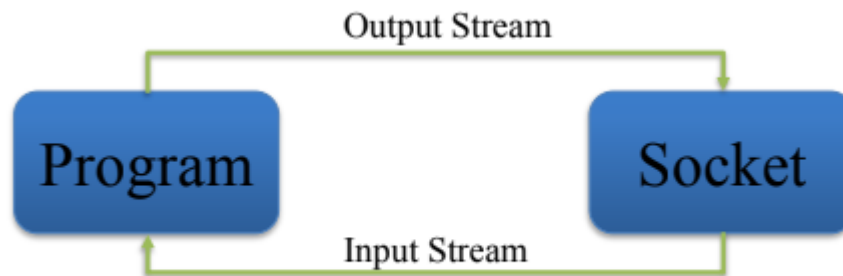
public InetAddress getLocalAddress()

Báo cho ta biết một Socket bị ràng buộc với giao diện mạng nào.

▪ Ví dụ:

```
Socket socket = new Socket(HostName, PortNumber);
InetAddress localAddress = socket.getLocalAddress( );
```

GetInputStream and GetOutputStream



public InputStream getInputStream() throws IOException

Trả về một dòng đầu vào có thể đọc dữ liệu từ socket vào một chương trình.

public OutputStream getOutputStream() throws IOException

Trả về dòng đầu ra của socket. Output stream được kết nối với input stream của socket remote.

c. Closing the Socket

```

public void close( ) throws IOException
Socket connection = null;
try {

connection = new Socket("[www.oreilly.com](http://www.oreilly.com/)", 13);
// tương tác với socket...

} // end try
catch (UnknownHostException ex) {
System.err.println(ex);
}
catch (IOException ex) {
System.err.println(ex);
}
finally {
if (connection != null) connection.close( ); }
  
```

public boolean isClosed()

Trả về giá trị true nếu socket đã đóng, trả về giá trị false nếu không

public boolean isConnected()

Phương thức isConnected () của lớp Java Socket trả về trạng thái kết nối của socket. Phương thức này sẽ tiếp tục trả về trạng thái kết nối trước khi bị đóng.

Half-closed sockets

Khi chúng ta truyền dữ liệu lên máy chủ và không biết lượng dữ liệu truyền đi là bao nhiêu, nếu chúng ta đóng socket khi truyền dữ liệu thì chúng ta sẽ bị ngắt kết nối với máy chủ ngay lập tức. Tại thời điểm này, chúng ta sẽ không thể nhận được phản hồi từ máy chủ.

Trong trường hợp này chúng ta sẽ Sử dụng phương pháp half close , nghĩa là chỉ đóng luồng đầu ra của một ổ cắm để cho biết rằng dữ liệu yêu cầu được gửi đến máy chủ đã kết thúc, nhưng luồng đầu vào phải được giữ ở trạng thái mở

1. public void shutdownInput() throws IOException
2. public void shutdownOutput() throws IOException

```
Socket connection = null;
try { connection = new Socket("[www.oreilly.com](http://www.oreilly.com/)", 80);
Writer out = new OutputStreamWriter(connection.getOutputStream( ), "8859_1");
out.write("GET / HTTP 1.0\r\n\r\n"); out.flush( );
connection.shutdownOutput( ); // Đọc câu trả lời...
} catch (IOException ex) {}
finally {
try {
if (connection != null) connection.close( ); }
catch (IOException ex) {}
}
```

d. Tùy chọn Socket

TCP_NODELAY

✓ public void setTcpNoDelay(boolean on) throws SocketException

✓ public boolean getTcpNoDelay() throws SocketException

- Nó sẽ Vô hiệu hóa thuật toán của Nagle.
- Hợp lệ cho client Sockets.

SO_LINGER

✓ public void setSoLinger(boolean on, int seconds) throws SocketException

✓ public int getSoLinger() throws SocketException

- Chỉ định thời gian chờ kéo dài khi đóng cửa.
- Hợp lệ cho client Sockets

SO_TIMEOUT

✓ public void setSoTimeout(int milliseconds) throws SocketException

✓ public int getSoTimeout() throws SocketException

- public String toString()
- Phương pháp toString () tạo ra một chuỗi như sau:

Socket[[addr=www.oreilly.com/198.112.208.11,port=80,localport=50055]](http://addr%3Dwww.oreilly.com/198.112.208.11,port=80,localport=50055)]

Socket[[addr=www.oreilly.com/198.112.208.11,port=80,localport=50055]](<http://addr%3Dwww.oreilly.com/198.112.208.11,port=80,localport=50055>)]

- Chỉ định thời gian chờ cho các hoạt động của socket chặn.
- Có giá trị cho tất cả các socket: Socket, ServerSocket, DatagramSocket.

1.1.3 Socket Exceptions

Hầu hết các phương pháp của lớp Socket được tuyên bố là throw IOException hoặc lớp con của nó, java.net.SocketException

Public class SocketException extends IOException

Phân lớp của SocketException

- **Public class BindException extends SocketException**
- **Public class ConnectException extends SocketException**

▪ **Public class NoRouteToHostException extends SocketException**

1.1.4 Socket Addresses

Trong thực tế, "socket" thường dùng để chỉ một socket trong mạng Giao thức Internet (IP) (nơi các socket có thể được gọi là Internet sockets), đặc biệt đối với Giao thức điều khiển truyền (TCP), là một giao thức cho các kết nối một-một. . Trong ngữ cảnh này, các socket được giả định được liên kết với một địa chỉ socket cụ thể, cụ thể là địa chỉ IP và số cổng cho nút cục bộ và có một địa chỉ socket tương ứng tại nút ngoại (nút khác), bản thân nó có một địa chỉ liên kết socket, được sử dụng bởi quy trình ngoại. Liên kết một socket với một địa chỉ socket được gọi là ràng buộc.

Lưu ý rằng trong khi quy trình cục bộ có thể giao tiếp với quy trình ngoại bằng cách gửi hoặc nhận dữ liệu đến hoặc từ địa chỉ socket ngoại, nó không có quyền truy cập vào socket ngoại bằng chính nó, cũng như không thể sử dụng bộ mô tả socket ngoại vì cả hai đều là nội bộ đến nút ngoại. Ví dụ: trong kết nối giữa 10.20.30.40:4444 và 50.60.70.80:8888 (địa chỉ IP cục bộ: cổng cục bộ, địa chỉ IP nước ngoài: cổng ngoại), cũng sẽ có một socket liên kết ở mỗi đầu, tương ứng với đại diện nội bộ của kết nối bởi ngăn xếp giao thức trên nút đó, được tham chiếu cục bộ bởi bộ mô tả socket số, giả sử 317 ở một bên và 922 ở bên kia. Một quy trình trên nút 10.20.30.40 có thể yêu cầu giao tiếp với nút 50.60.70.80 trên cổng 8888 (yêu cầu ngăn xếp giao thức tạo một socket để giao tiếp với đích đó) và khi nó đã tạo một socket và nhận được một bộ mô tả socket (317) , nó có thể giao tiếp qua socket này bằng cách sử dụng bộ mô tả (317): ngăn xếp giao thức sau đó sẽ chuyển tiếp dữ liệu đến và từ nút 50.60.70.80 trên cổng 8888. Tuy nhiên, một quy trình trên nút 10.20.30.40 không thể yêu cầu giao tiếp với "socket 922" hoặc "socket 922 trên nút 50.60.70.80 ": đây là những con số vô nghĩa đối với ngăn xếp giao thức trên nút 10.20.30.40.

Lớp trừu tượng rỗng, không có phương pháp nào ngoài kiến tạo ngầm định như sau:

```
package java.net.*;
public abstract class SocketAddress {
    public SocketAddress( ) {}
}
```

Public SocketAddress getRemoteSocketAddress()

- Trả về địa chỉ của hệ thống đang được kết nối với

Public SocketAddress getLocalSocketAddress()

- Trả về địa chỉ nơi kết nối

A SocketAddress

- Public void connect(SocketAddress endpoint) throws IOException

Ví dụ, đầu tiên kết nối với Yahoo, sau đó lưu trữ địa chỉ của nó:

```
Socket socket = new Socket("www.yahoo.com", 80);
SocketAddress yahoo = socket.getRemoteSocketAddress( );
socket.close( );
```

Sau đó, kết nối lại với Yahoo bằng cách sử dụng địa chỉ này:

```
socket = new Socket( );
socket.connect(yahoo);
```

Các bước tạo Client

Bước 1: tạo Socket

```
Socket socket = new Socket(serverName, portNumber);
```

Bước 2: Tạo InputStream

```
InputStream inputStream = socket.getInputStream();
Scanner scanner = new Scanner(inputStream);
```

Bước 3: Tạo OutputStream

```
OutputStream outputStream = socket.getOutputStream();
PrintWriter printWriter = new PrintWriter(outPutStream, true)
```

Bước 4: Gửi và nhận dữ liệu

```
String strReceive = scanner.nextLine(); //Nhận dữ liệu từ Server
printWriter.println(sending data); //Gửi dữ liệu đến Sever
```

1.2 Socket for server

1.2.1 The Constructors

Bốn constructor public ServerSocket:

Public ServerSocket(int port) throws BindException, IOException

Constructor này tạo một socket cho server trên cổng xác định. Nếu port bằng 0, hệ thống chọn một cổng ngẫu nhiên cho ta. Cổng do hệ thống chọn đôi khi được gọi là cổng vô danh vì ta không biết số hiệu cổng. Với các server, các cổng vô danh không hữu ích lắm vì các client cần phải biết trước cổng nào mà nó nối tới (giống như người gọi điện thoại ngoài việc xác định cần gọi cho ai cần phải biết số điện thoại để liên lạc với người đó).

Ví dụ:

```
try {
    ServerSocket httpd = new ServerSocket(80);
}catch (IOException ex) {
    System.err.println(ex);
}
```

Public ServerSocket(int port, int queueLength) throws IOException

Mở socket máy chủ trên cổng được chỉ định với độ dài hàng đợi người dùng chọn

Ví dụ:

```
try {
    ServerSocket httpd = new ServerSocket(5776,100);
}catch (IOException ex) {
    System.err.println(ex);
}
```

**Public ServerSocket(int port, int queueLength, InetAddress bindAddress)
throws IOException**

Constructor này tạo một đối tượng ServerSocket trên cổng xác định với chiều dài hàng đợi xác định. ServerSocket chỉ gán cho địa chỉ IP cục bộ xác định. Constructor này hữu ích cho các server chạy trên các hệ thống có nhiều địa chỉ IP.

Public ServerSocket() throws IOException

Ví dụ:

```

ServerSocket ss = new ServerSocket( );
// set socket options...
SocketAddress http = new InetSocketAddress(80);
ss.bind(http);

```

1.2.2 Accepting and Closing Connections

Accepting

Với một socket được đánh dấu là đang lắng nghe, máy chủ có thể chấp nhận kết nối:

```

struct sockaddr_in from;
...
fromlen = sizeof (from);
newsock = accept (s, (struct sockaddr *) & from, & fromlen);

```

Khi một kết nối được chấp nhận, một bộ mô tả mới sẽ được trả về (cùng với một socket mới). Một đối tượng `ServerSocket` hoạt động trong một vòng lặp chấp nhận các liên kết. Mỗi lần lặp nó gọi phương thức `accept()`. Phương thức này trả về một đối tượng `Socket` biểu diễn liên kết giữa client và server. Tương tác giữ client và server được tiến hành thông qua socket này. Khi giao tác hoàn thành, server gọi phương thức `close()` của đối tượng socket. Nếu client ngắt liên kết trong khi server vẫn đang hoạt động, các luồng vào ra kết nối server với client sẽ đưa ra ngoại lệ `InterruptedException` trong lần lặp tiếp theo. Khi bước thiết lập liên kết hoàn thành, và ta sẵn sàng để chấp nhận liên kết, cần gọi phương thức `accept()` của lớp `ServerSocket`. Phương thức này phong tỏa; nó dừng quá trình xử lý và đợi cho tới khi client được kết nối. Khi client thực sự kết nối, phương

thức `accept()` trả về đối tượng `Socket`. Ta sử dụng các phương thức `getInputStream()` và `getOutputStream()` để truyền tin với client.

▪ **Public `Socket accept()` throws `IOException`**

Ví dụ:

```
try { ServerSocket server = new ServerSocket(5776);
while (true) { Socket connection = server.accept( );
try { Writer out = new OutputStreamWriter(connection.getOutputStream( ));
out.write("You've connected to this server. Bye-bye now.\r\n");
out.flush( );
connection.close( );
} catch (IOException ex) {
} finally { // Guarantee that sockets are closed when complete.
try { if (connection != null) connection.close( ); }
catch (IOException ex) {}
}
} catch (IOException ex) {
System.err.println(ex); }
```

Closing

Khi một socket không còn được quan tâm, nó có thể bị loại bỏ bằng cách áp dụng một giá trị đóng vào bộ mô tả:

```
close (s);
```

Nếu dữ liệu được liên kết với một socket hứa hẹn phân phối đáng tin cậy (ví dụ: socket luồng) khi `close()` diễn ra, hệ thống sẽ tiếp tục cố gắng truyền dữ liệu. Tuy nhiên, nếu sau một khoảng thời gian khá dài mà dữ liệu vẫn không được phân phối, nó sẽ bị loại bỏ. Nếu bạn không sử dụng bất kỳ dữ liệu đang chờ xử lý nào, bạn có thể thực hiện `shutdown()` trên socket trước khi đóng nó:

```
shutdown(s, how);
```

Trong đó how là 0 nếu bạn không còn quan tâm đến việc đọc dữ liệu, là 1 nếu không còn dữ liệu nào được gửi hoặc là 2 nếu không có dữ liệu nào được gửi hoặc nhận.

Nếu đã hoàn thành công việc với một ServerSocket, ta cần phải đóng nó lại, đặc biệt nếu chương trình của ta tiếp tục chạy. Điều này nhằm tạo điều kiện cho các chương trình khác muốn sử dụng nó. Đóng một ServerSocket không đồng nhất với việc đóng một Socket. Lớp ServerSocket cung cấp một số phương thức cho ta biết địa chỉ cục bộ và cổng mà trên đó đối tượng server đang hoạt động. Các phương thức này hữu ích khi ta đã mở một đối tượng server socket trên một cổng vô danh và trên một giao tiếp mạng không

Public void close() throws IOException

Ví dụ:

```
for (int port = 1; port <= 65535; port++) {
    try {
        ServerSocket server = new ServerSocket(port);
        server.close( );
    }
    catch (IOException ex) {
        System.out.println("There is a server on port " + port + ".");
    }
} // end for
```

1.2.3 The get Methods

Public InetAddress getInetAddress()

Phương thức này trả về địa chỉ được sử dụng bởi server (localhost). Nếu localhost

có địa chỉ IP, địa chỉ này được trả về bởi phương thức `InetAddress.getLocalHost()`

Ví dụ:

```
try{
    ServerSocket httpd = new ServerSocket(80);
    InetAddress ia = httpd.getInetAddress( );
}catch(IOException e) {}
```

Public int getLocalPort()

Các constructor `ServerSocket` cho phép ta nghe dữ liệu trên cổng không định trước bằng cách gán số 0 cho cổng. Phương thức này cho phép ta tìm ra cổng mà server đang nghe.

Ví dụ:

```
try {
    ServerSocket server = new ServerSocket(0);
    System.out.println("This server runs on port "+ server.getLocalPort( ));
}
catch (IOException ex) {
    System.err.println(ex);
}
```

1.2.4 Socket Options

Các socket option này có thể ảnh hưởng đến hành vi của Sockets layer, Transport layer, và IP layer. Việc sử dụng các socket option phổ biến nhất là tăng hiệu suất của kết nối, nhưng chúng cũng được sử dụng rộng rãi trong các tình huống khác. Chúng ta xem xét rất nhiều socket option có sẵn và cung cấp các ví dụ cho mỗi socket option.

Vị trí	Tên Option	Kiểu dữ liệu	Get /Set	Mô tả Option
SOL_SOCKET	SO_ACCEPTCONN	BOOL	Get	Socket có lắng nghe không?
	SO_BROADCAST	BOOL	Get/Set	Cho phép phát tin nhắn broadcast trên socket.
	SO_DONTLINGER	BOOL	Get/Set	Cho phép hoặc vô hiệu hóa trả về ngay lập tức từ closeSocket ()
	SO_KEEPALIVE	BOOL	Get/Set	Gửi tin nhắn keep-alive
	SO_LINGER	struct linger	Get/Set	Cho phép hoặc vô hiệu hóa trả về ngay lập tức từ closeSocket ()
	SO_OOBINLINE	BOOL	Get/Set	Out-of-band data nằm trong data stream thông thường
	SO_REUSEADDR	BOOL	Get/Set	Cho phép hoặc vô hiệu hóa việc tái sử dụng một socket bị ràng buộc
	SO_SECURE	DWORD	Get/Set	Cho phép hoặc vô hiệu hóa mã hóa SSL trên socket
	SO_SNDBUF	Int	Get/Set	Kích thước của bộ đệm được phân bổ để gửi dữ liệu
	SO_TYPE	Int	Get	Kiểu socket

IPPROTO_TCP	TCP_NODELAY	BOOL	Get/Set	Bật/Tắt thuật toán NAGLE
IPPROTO_IP	IP_MULTICAST_TTL	Int	Get/Set	Thời gian để duy trì một gói phát đa hướng
	IP_MULTICAST_IF	unsigned long	Get/Set	Địa chỉ giao diện phát đa hướng đi
	IP_ADD_MEMBERSHIP	struct ip_mreq	Get/Set	Thêm socket vào một nhóm phát đa hướng
	IP_DROP_MEMBERSHIP	struct	Get/Set	Loại bỏ socket từ một nhóm phát đa hướng

Bảng 1.2.4 : Socket Option

1.2.5 The Object Methods

STT	Tên phương thức	Mô tả
1	public Socket accept() throws IOException	Phương thức này chặn cho đến khi hoặc một client kết nối với server trên cổng được chỉ định hoặc socket time-out giả định rằng giá trị time-out được thiết lập theo phương thức setSoTimeout(). Nếu không thì phương thức này chặn vô thời hạn.

2	<code>public InetAddress getInetAddress()</code>	Phương thức này trả về địa chỉ của máy tính khác mà socket này được kết nối.
3	<code>public int getPort()</code>	Trả về cổng socket được liên kết với máy từ xa.
4	<code>public int getLocalPort()</code>	Trả về cổng socket được liên kết với máy cục bộ.
5	<code>Public SocketAddress getRemoteSocketAddress()</code>	Trả về địa chỉ của socket từ xa.
6	<code>public InputStream getInputStream() throws IOException</code>	Trả về input stream của socket. Input stream được kết nối với output stream của socket từ xa.
7	<code>public OutputStream getOutputStream() throws IOException</code>	Trả về output stream của socket. Output stream được kết nối với luồng đầu vào của socket từ xa.
8	<code>public void close() throws IOException</code>	Đóng socket, giúp socket object này không còn có khả năng kết nối lại với bất kỳ server nào.
9	<code>public void setSocketTimeot(int timeout)</code>	Đặt giá trị time-out cho server socket trong bao lâu để chờ client accept().

Bảng 1.2.5 : Object Method

1.2.6 Implementation

Bước 1: Tạo một đối tượng ServerSocket

```
ServerSocket ss=new ServerSocket(port)
```

Bước 2: Tạo một đối tượng Socket bằng cách chấp nhận liên kết từ yêu cầu liên kết của client. Sau khi chấp nhận liên kết, phương thức `accept()` trả về đối tượng Socket thể hiện liên kết giữa Client và Server.

```
while(condion) {
    Socket s=ss.accept();
    doSomething(s);
}
```

Bước 3: Tạo một luồng nhập để đọc dữ liệu từ client.

```
BufferedReader in=new BufferedReader(new InputStreamReader(s.getInputStream()));
```

Bước 4: Tạo một luồng xuất để gửi dữ liệu trở lại cho server.

```
PrintWriter pw=new PrintWriter(s.getOutputStream(),true);
//Trong đó tham số true được sử dụng để xác định rằng luồng sẽ được tự động đẩy ra.
```

Bước 5: Thực hiện các thao tác vào ra với các luồng nhập và luồng xuất.

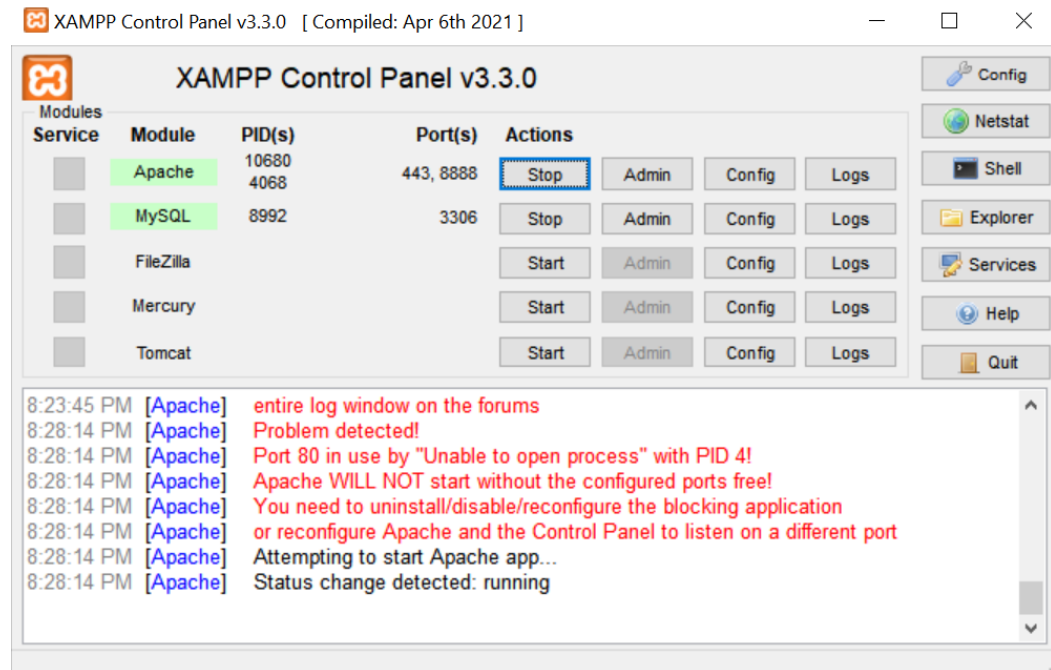
```
BufferedReader br=new BufferedReader(new InputStreamReader(s.getInputStream()));

while(true){
    String line=br.readLine();
    if(line.equals("exit"))break;
    String upper=line.toUpperCase();
    pw.println(upper);
    pw.flush();
}
```

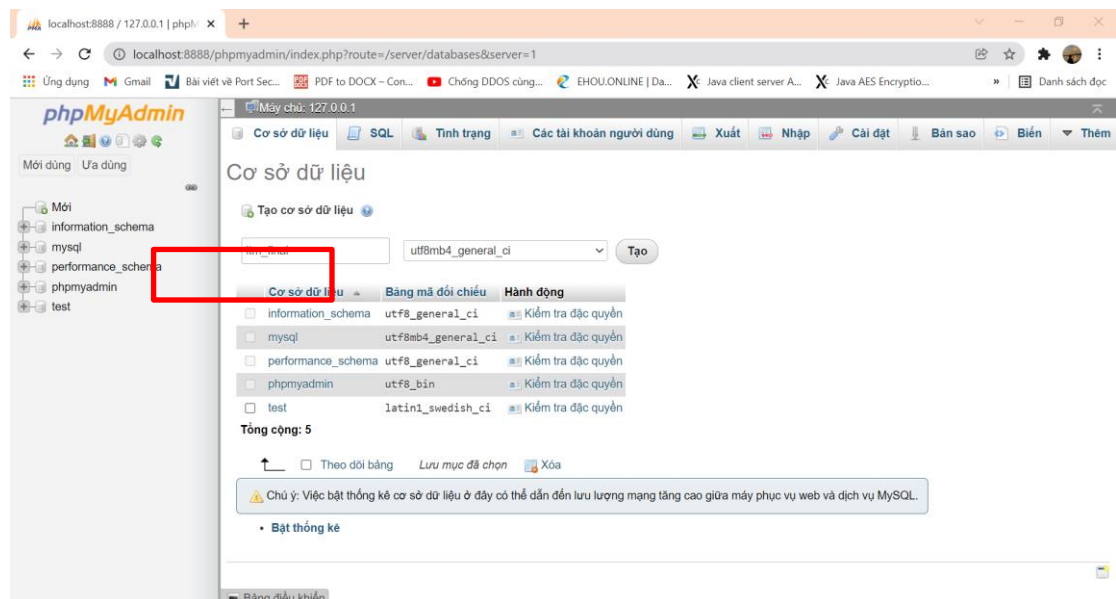
- Bước 6: Đóng socket `s` khi đã truyền tin xong. Việc đóng socket cũng đồng nghĩa với việc đóng các luồng.

```
finally{  
    try{  
        if(s!=null){  
            s.close();  
        }  
    } catch(IOException e){}  
}
```

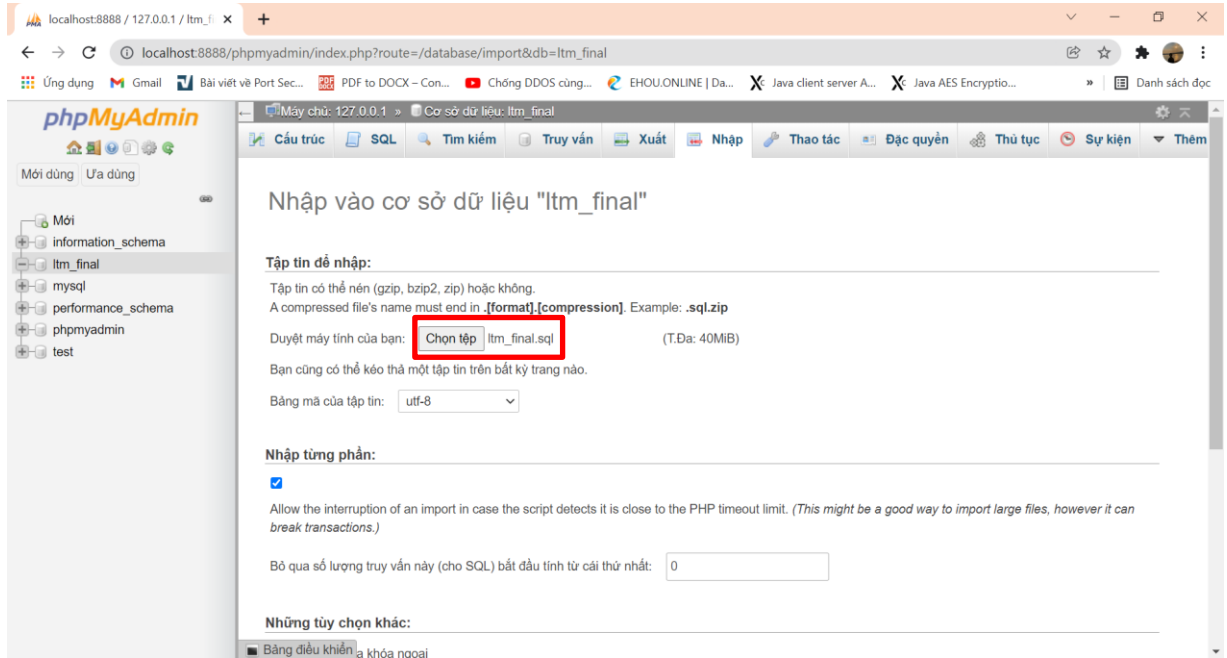
CHƯƠNG 2 _MÔ HÌNH DEMO



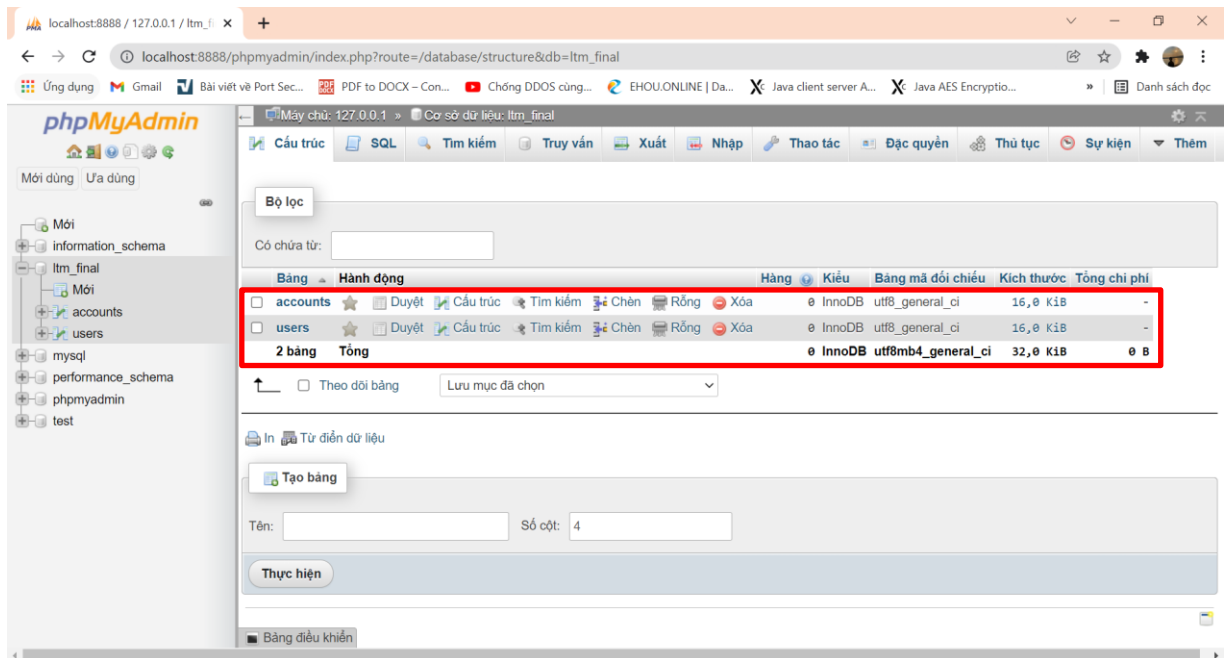
Hình 2.1 Mở XAMPP và nhấn “start” Apache và MySQL.



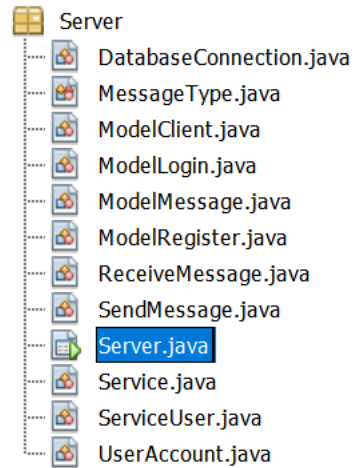
Hình 2.2 : Tạo cơ sở dữ liệu tên “ltn_final”.



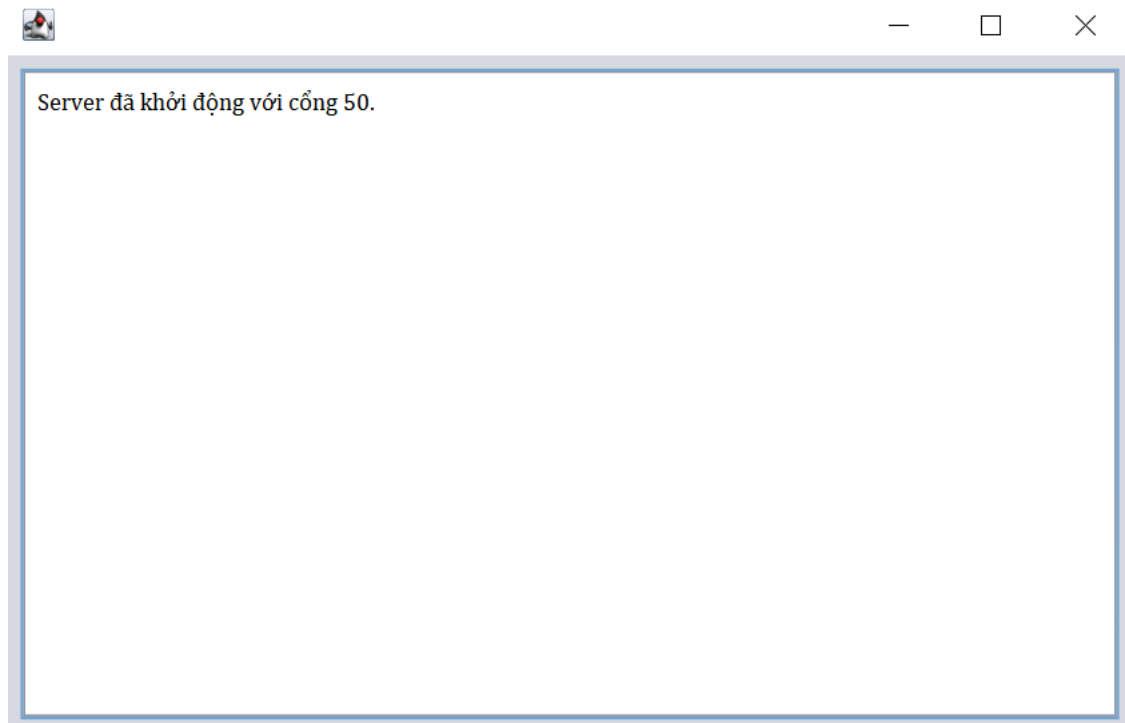
Hình 2.3 : Import file ltm_final.sql vào cơ sở dữ liệu ltm_final.



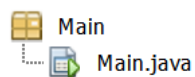
Hình 2.4 : Khởi tạo hai bảng “accounts” và “users” thành công.



Hình 2.5 Chạy file Server.java.



Hình 2.6 Giao diện file Server.java.



Hình 2.7 Chạy file Main.java.



ĐĂNG NHẬP



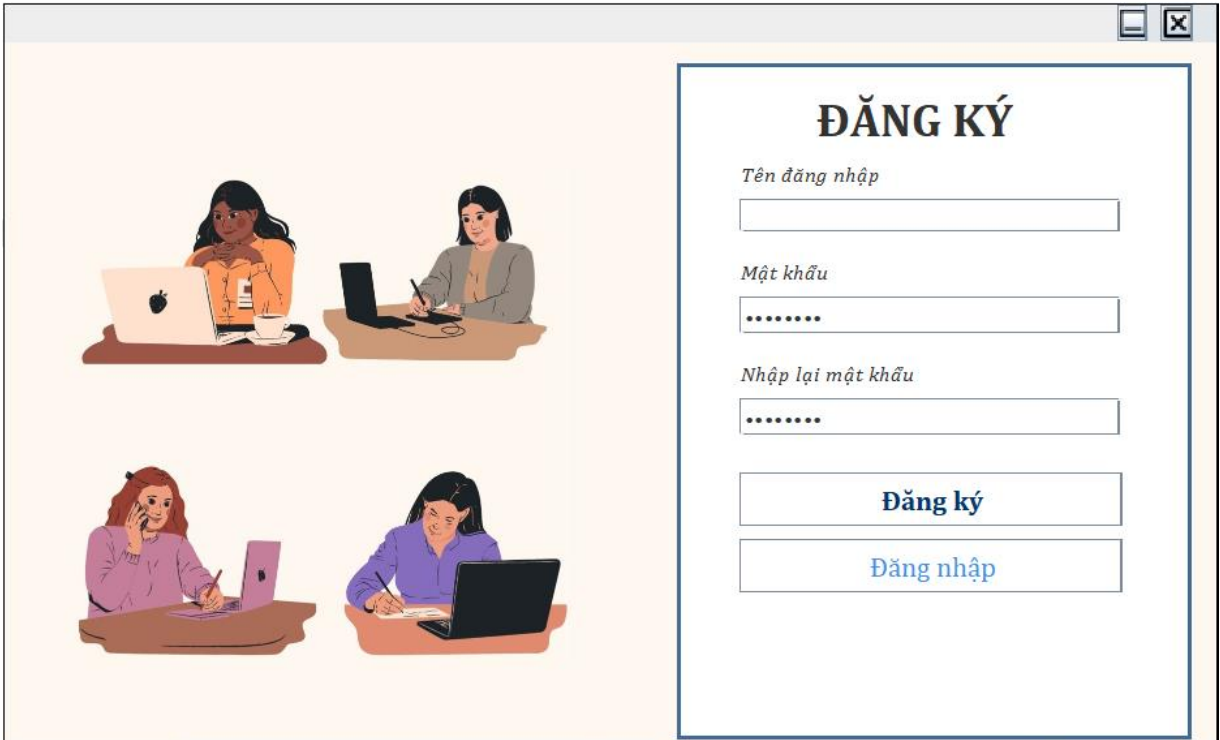
Tên đăng nhập

Mật khẩu

Đăng nhập

Đăng ký

Hình 2.8 : Giao diện đăng nhập của file Main.java.



ĐĂNG KÝ

Tên đăng nhập

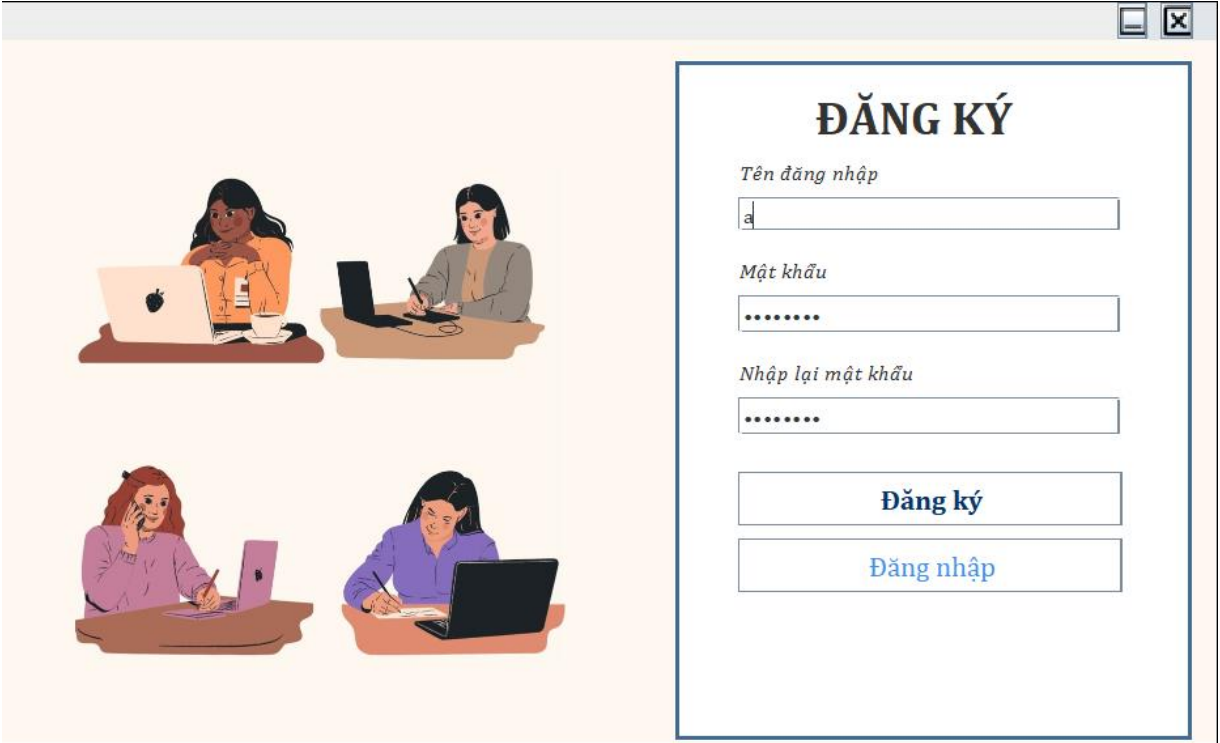
Mật khẩu

Nhập lại mật khẩu

Đăng ký

Đăng nhập

Hình 2.9 : Giao diện đăng ký tài khoản của file Main.java.



ĐĂNG KÝ

Tên đăng nhập

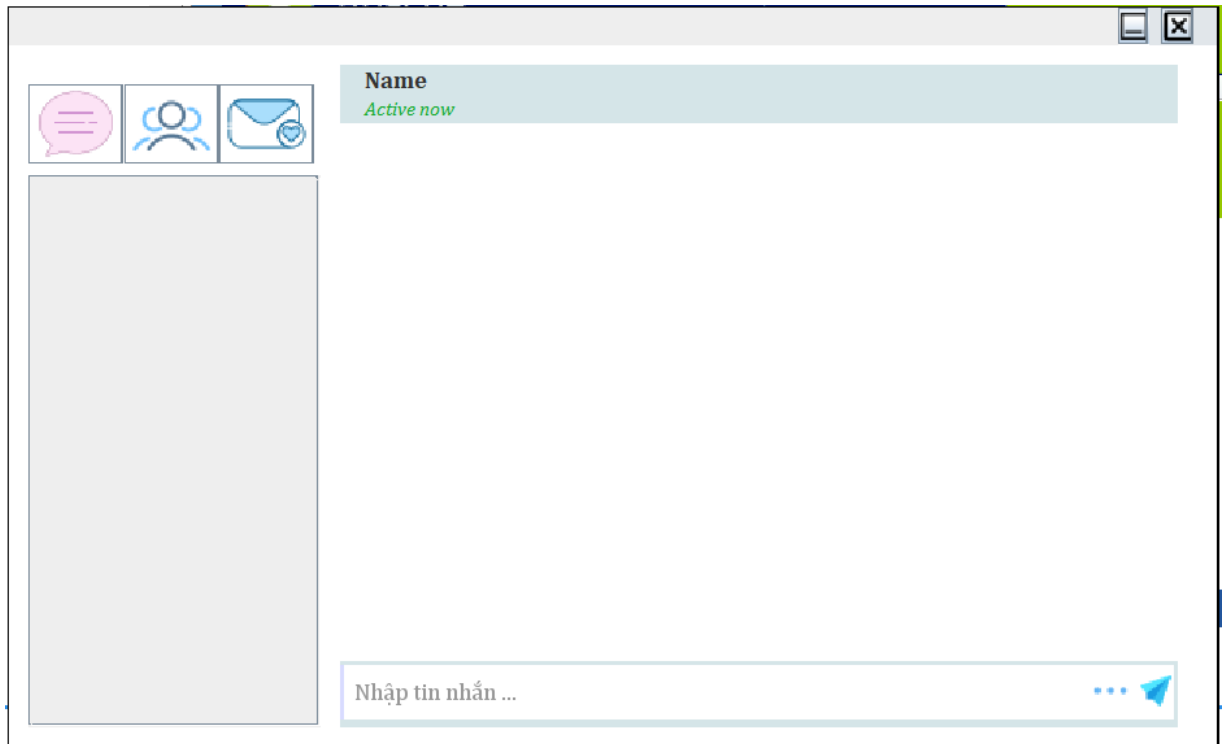
Mật khẩu

Nhập lại mật khẩu

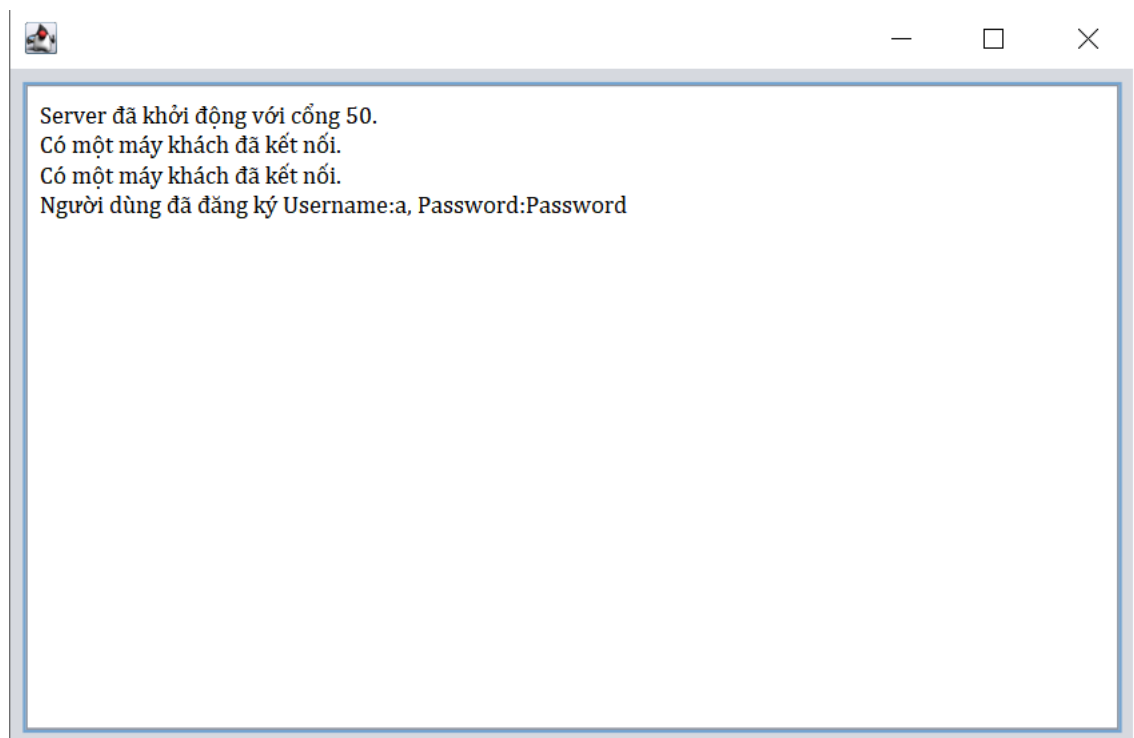
Đăng ký

Đăng nhập

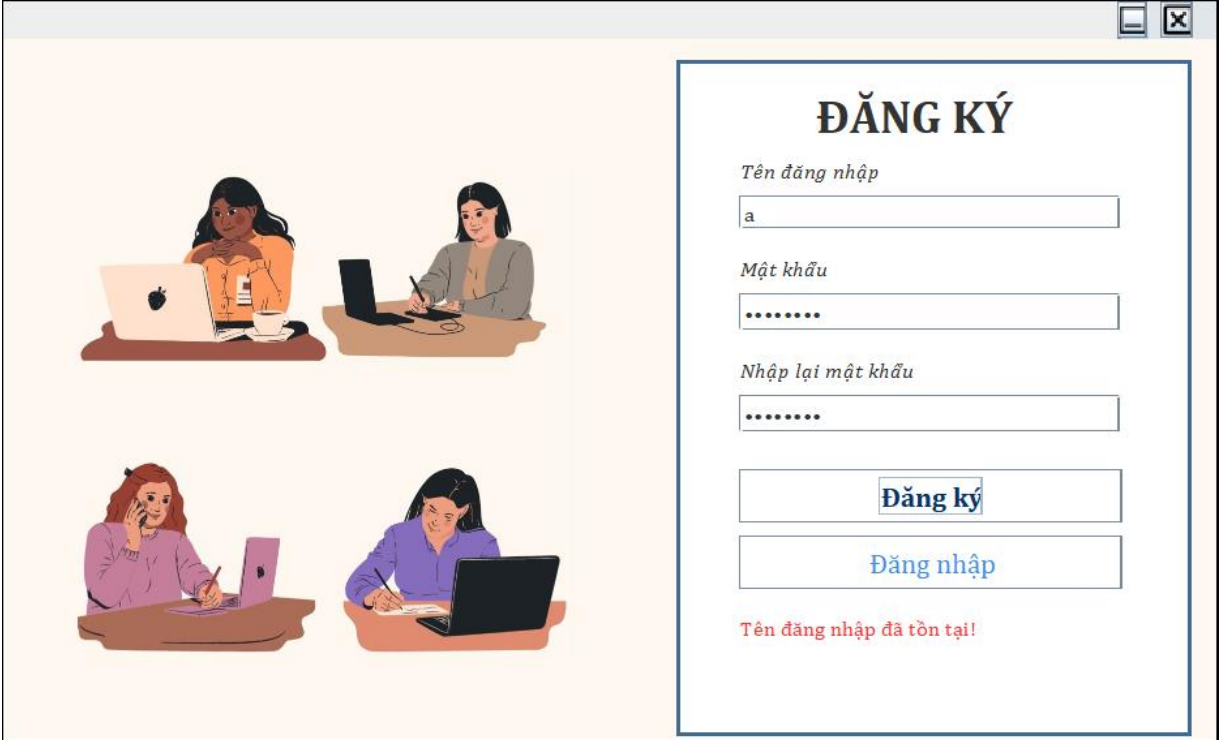
Hình 2.10 : Đăng ký tài khoản “a” với mật khẩu là “password”.



Hình 2.11 : Giao diện Chat khi đăng ký tài khoản thành công.




Hình 2.12 : Server sẽ hiển thị các thông báo khi có các máy khách kết nối với server và các thông tin đăng ký tài khoản.



The screenshot displays a web application interface. On the left, there is an illustration of four people (two women and two men) sitting at desks, each working on a laptop. On the right, there is a registration form titled "ĐĂNG KÝ". The form contains the following fields and elements:

- Tên đăng nhập** (Login Name): A text input field containing the letter "a".
- Mật khẩu** (Password): A password input field with masked characters (dots).
- Nhập lại mật khẩu** (Repeat Password): A password input field with masked characters (dots).
- Đăng ký** (Register): A button with the text "Đăng ký".
- Đăng nhập** (Login): A button with the text "Đăng nhập".
- Tên đăng nhập đã tồn tại!** (Login name already exists!): A red error message displayed below the registration button.

Hình 2.13 : Hệ thống Chat sẽ hiển thị thông báo tên đăng nhập đã tồn tại để người dùng chọn tên đăng nhập khác để đăng ký tài khoản.



The illustration shows four people working on laptops. In the top left, a woman with dark hair and an orange shirt is looking at a laptop. In the top right, a woman with dark hair and a grey shirt is writing on a notepad. In the bottom left, a woman with red hair and a purple shirt is talking on a phone while looking at a laptop. In the bottom right, a woman with dark hair and a purple shirt is writing on a notepad while looking at a laptop.

ĐĂNG KÝ

Tên đăng nhập

Mật khẩu

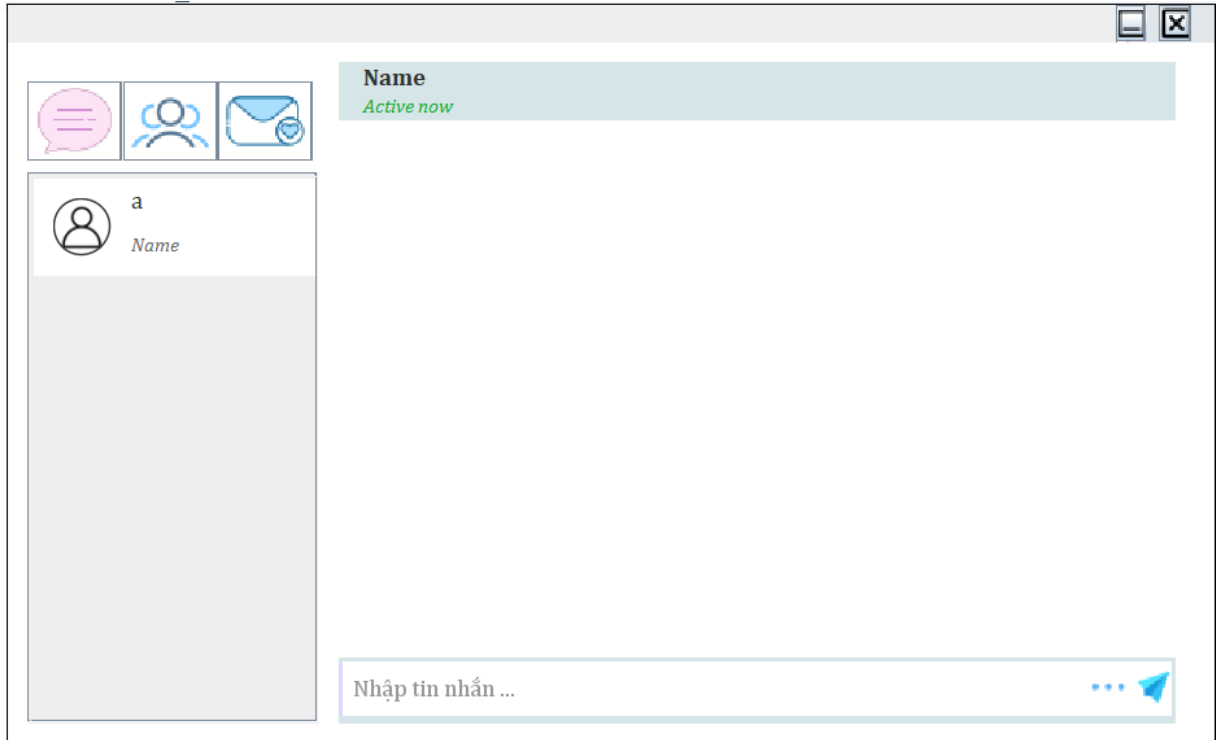
Nhập lại mật khẩu

Đăng ký

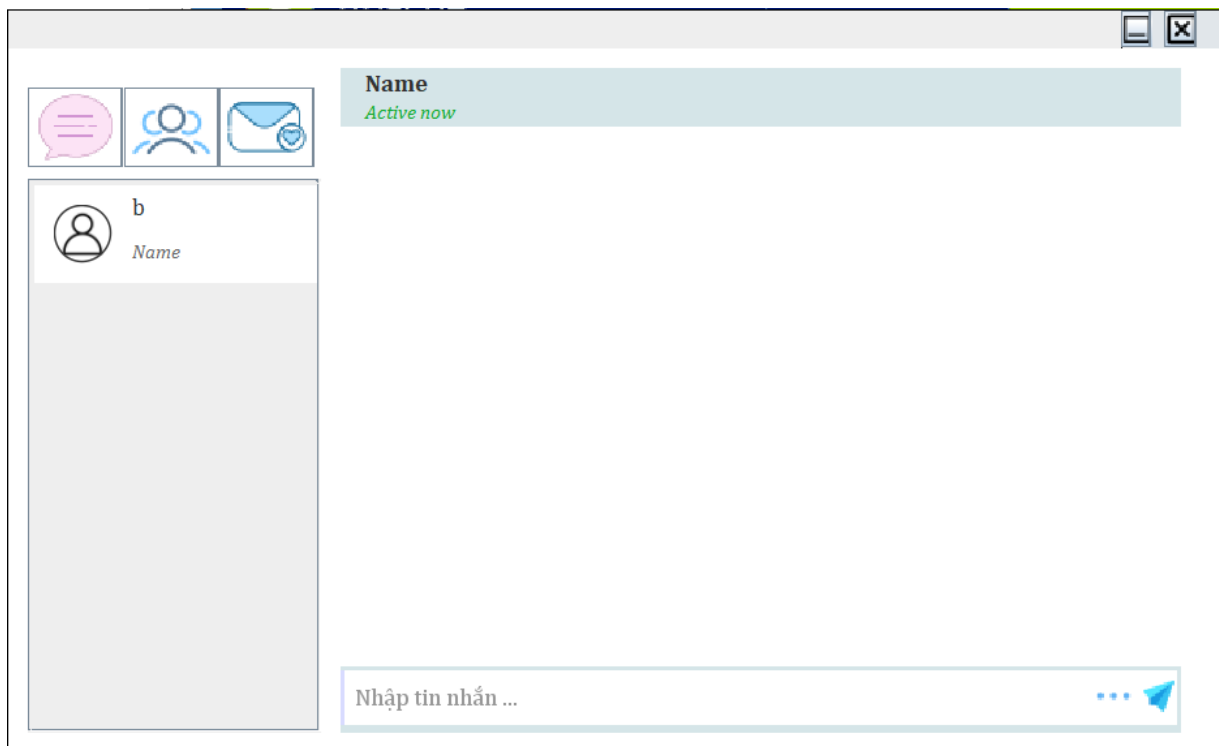
Đăng nhập

Tên đăng nhập đã tồn tại!

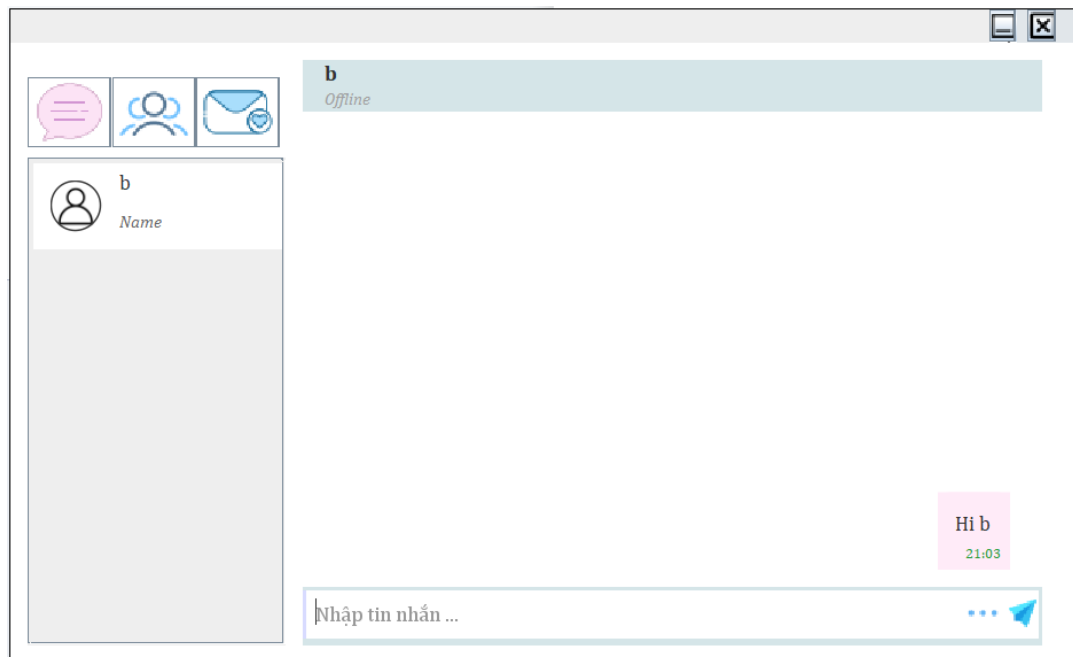
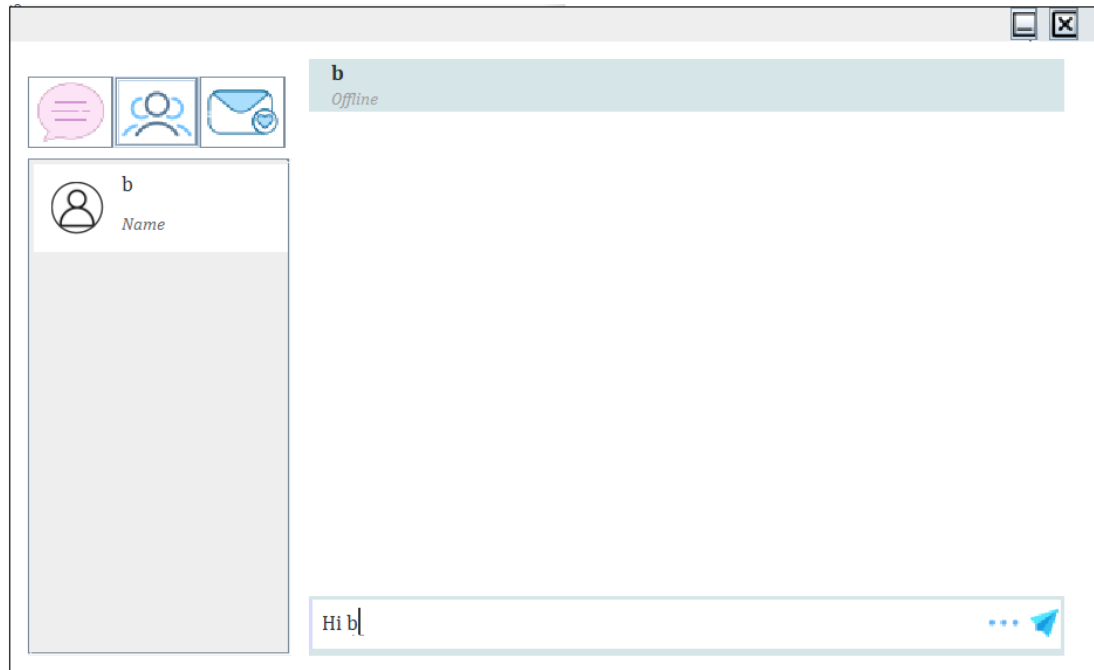
Hình 2.14 : Đăng ký một tài khoản khác với tên đăng nhập là “b” với mật khẩu là “password”.



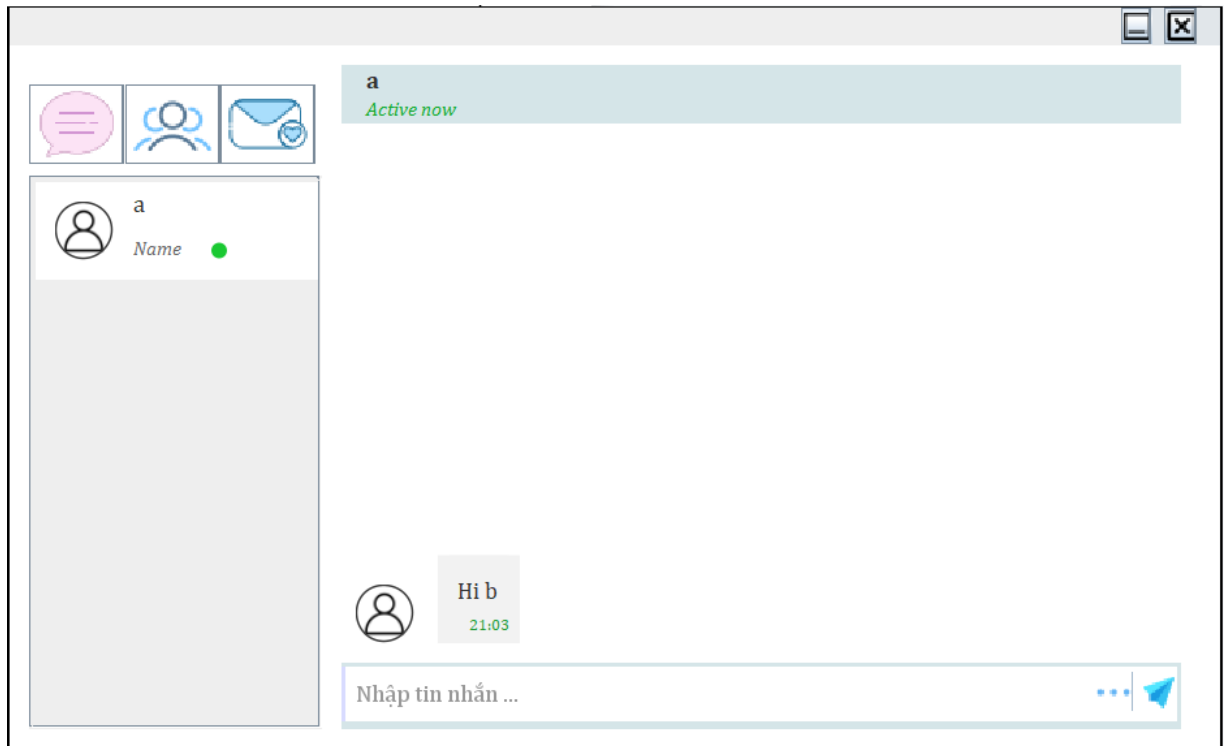
Hình 2.15 : Hệ thống chat sẽ hiển thị các tài khoản đã đăng ký trước đó ở giao diện Chat của người dùng “b”.



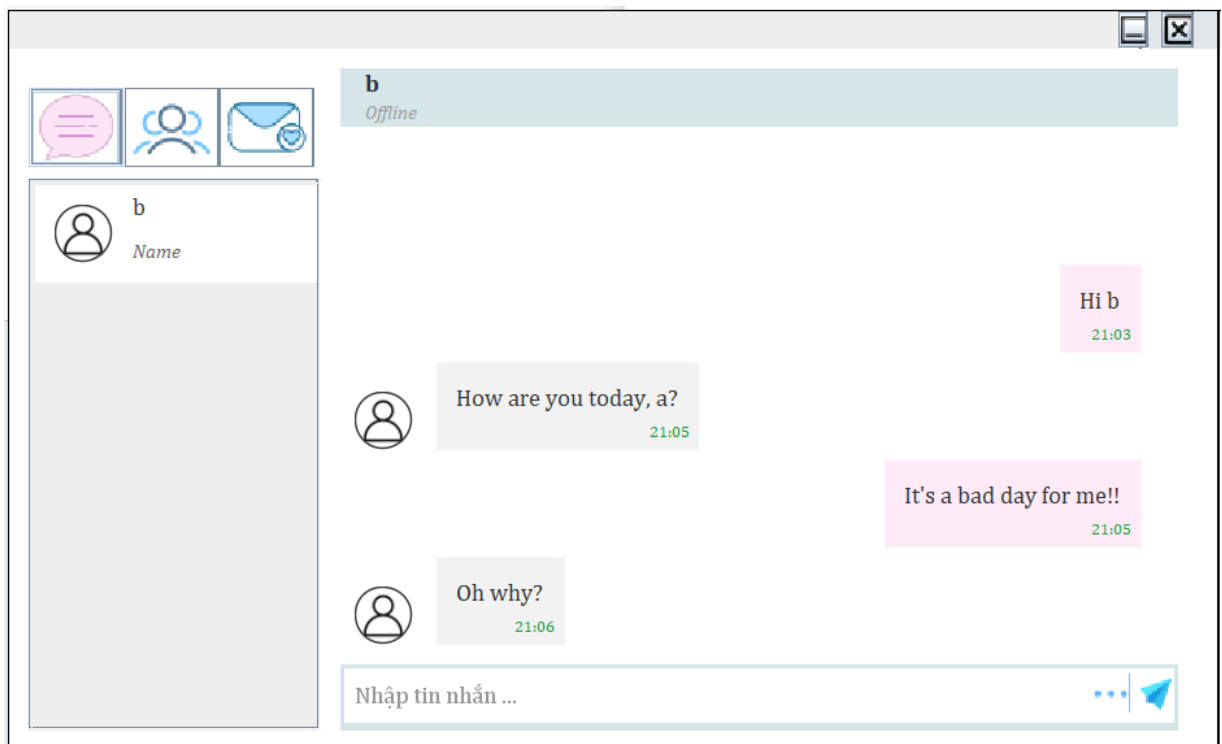
Hình 2.16: Khi này, giao diện của người dùng “a” cũng sẽ hiển thị tài khoản người dùng “b”.

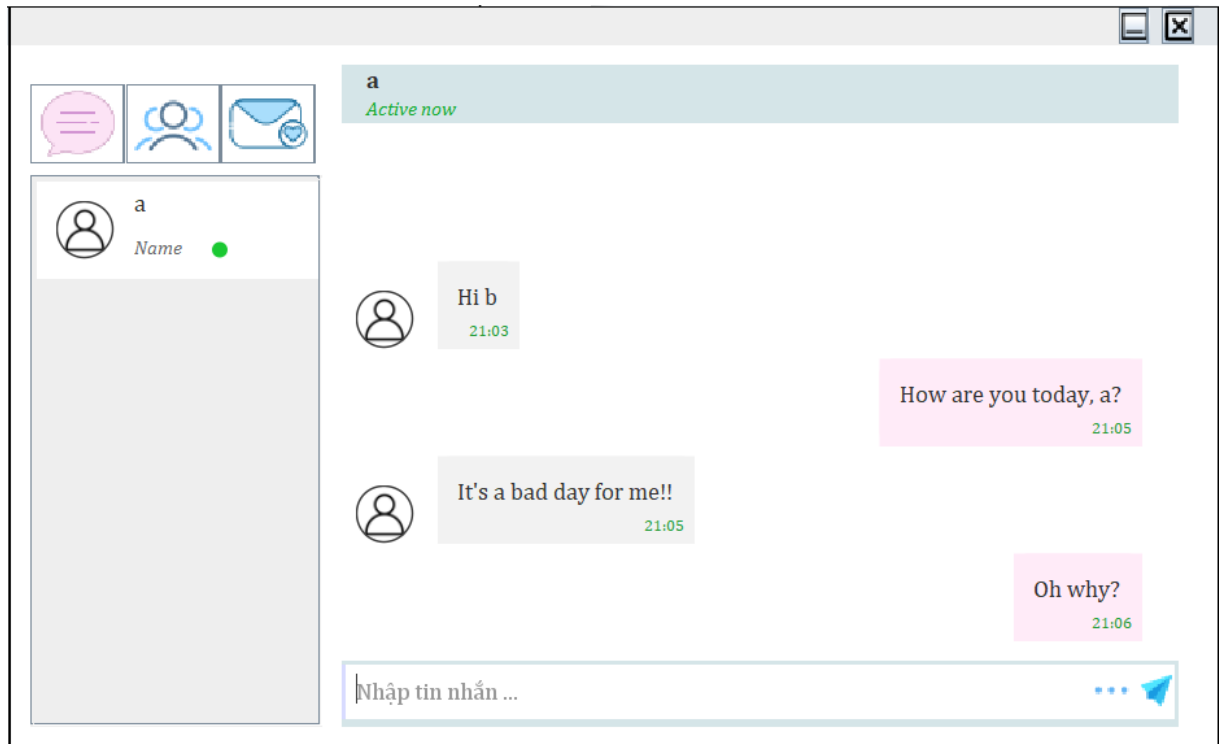


Hình 2.17 : Người dùng “a” thực hiện thao tác gửi tin nhắn cho người dùng “b”. Ngoài nội dung tin nhắn thì hệ thống Chat còn hiển thị thêm thời gian gửi tin nhắn đó.

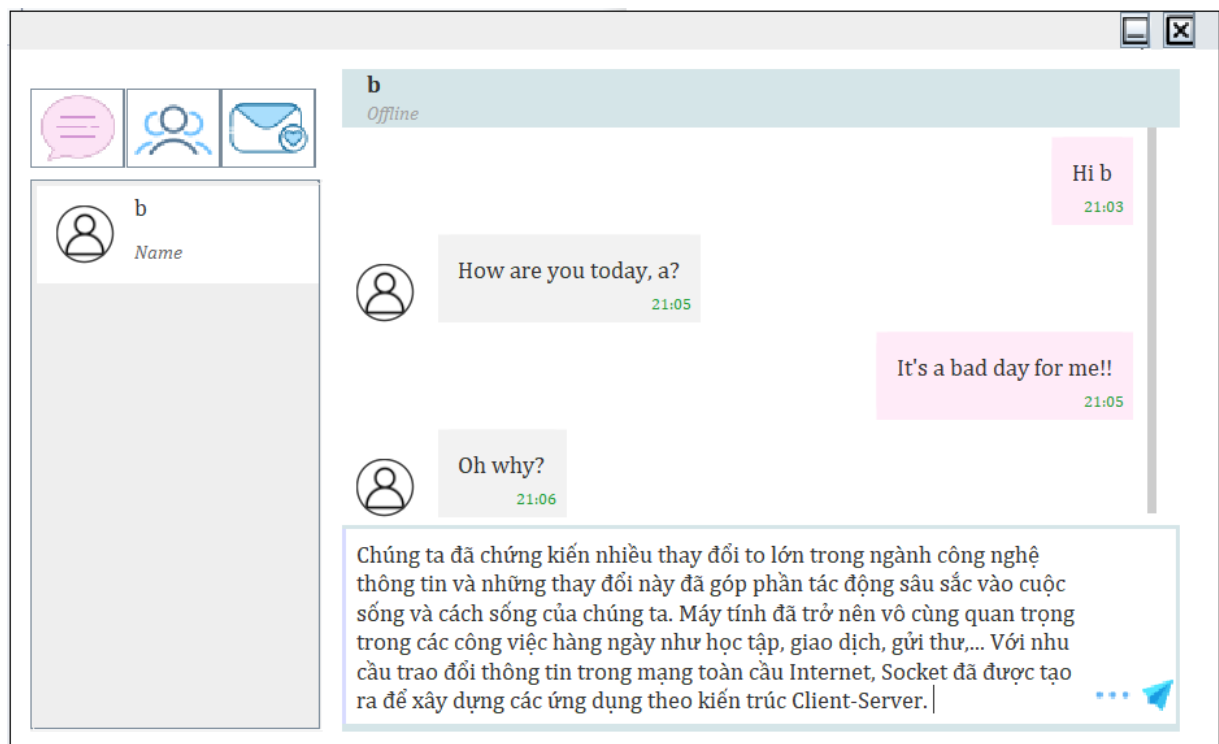


Hình 2.18 : Phía người dùng “b” nhận được tin nhắn từ người dùng “a”.

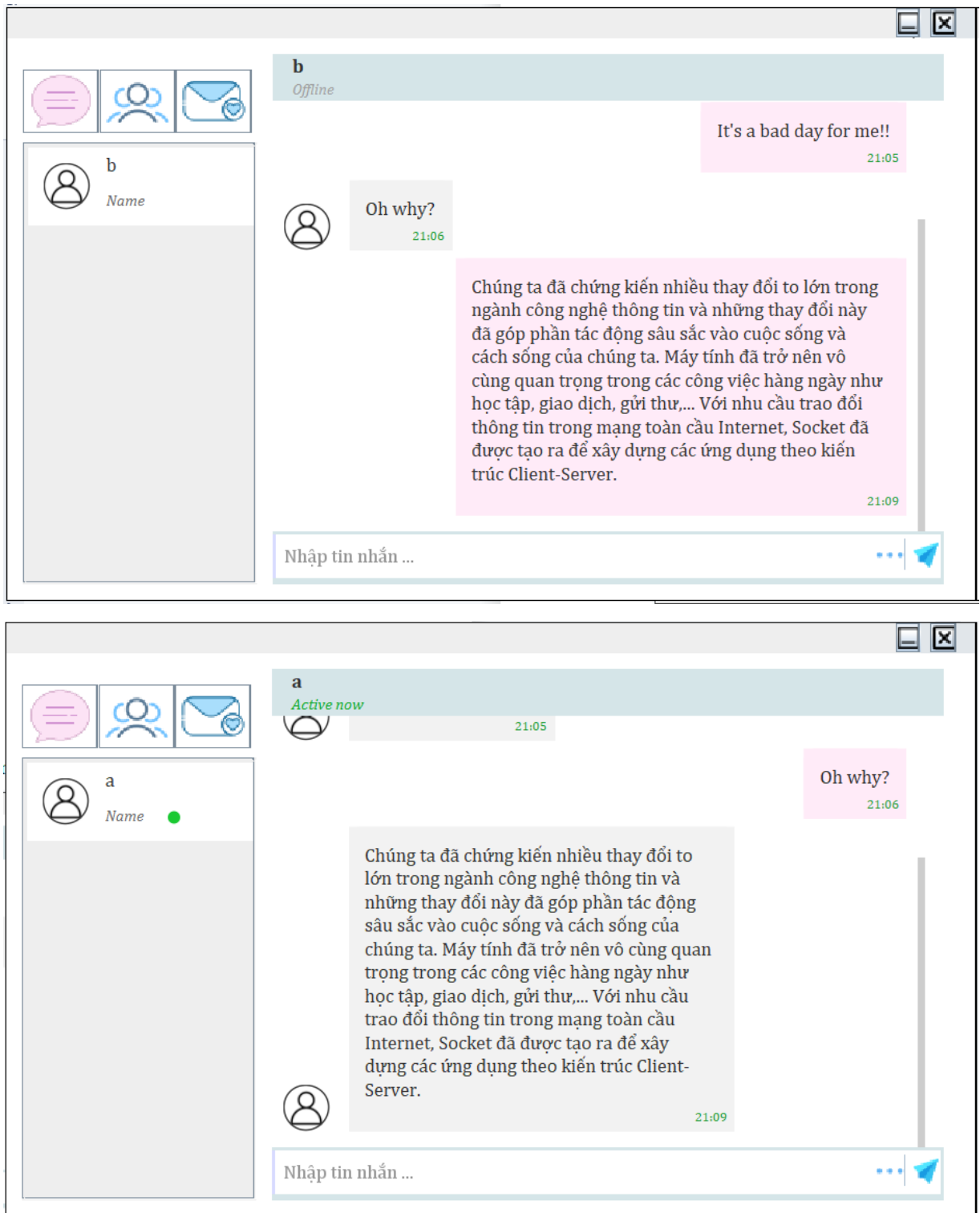




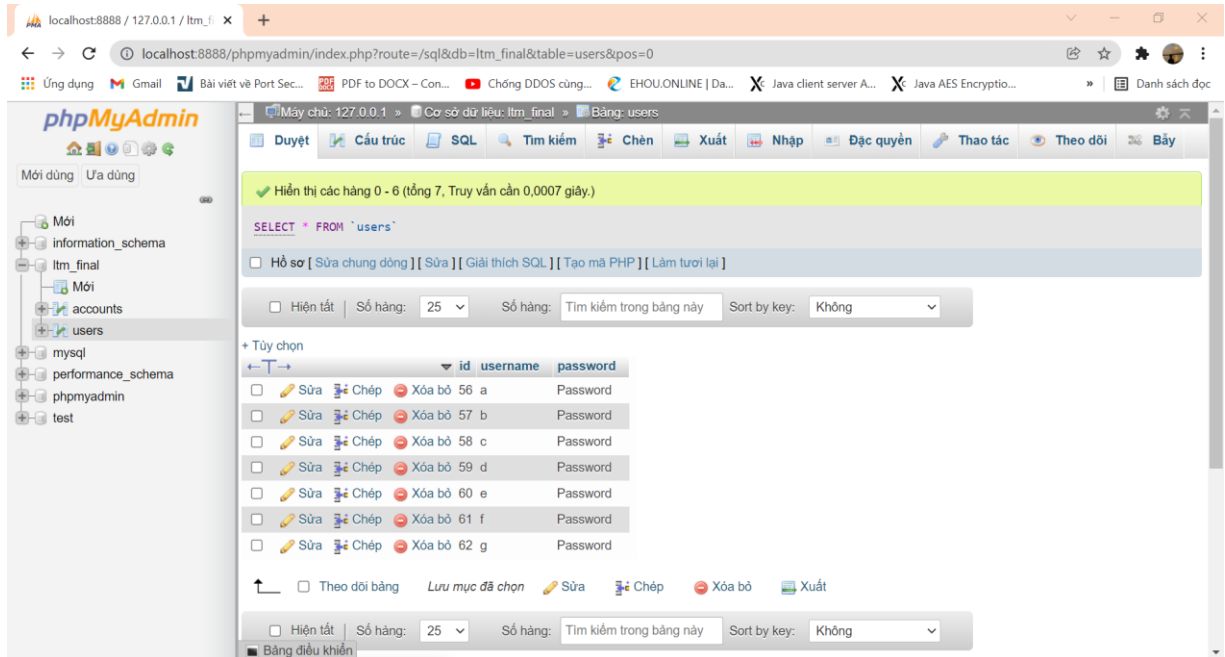
Hình 2.19 : Đoạn tin nhắn giữa người dùng “a” và người dùng “b”.



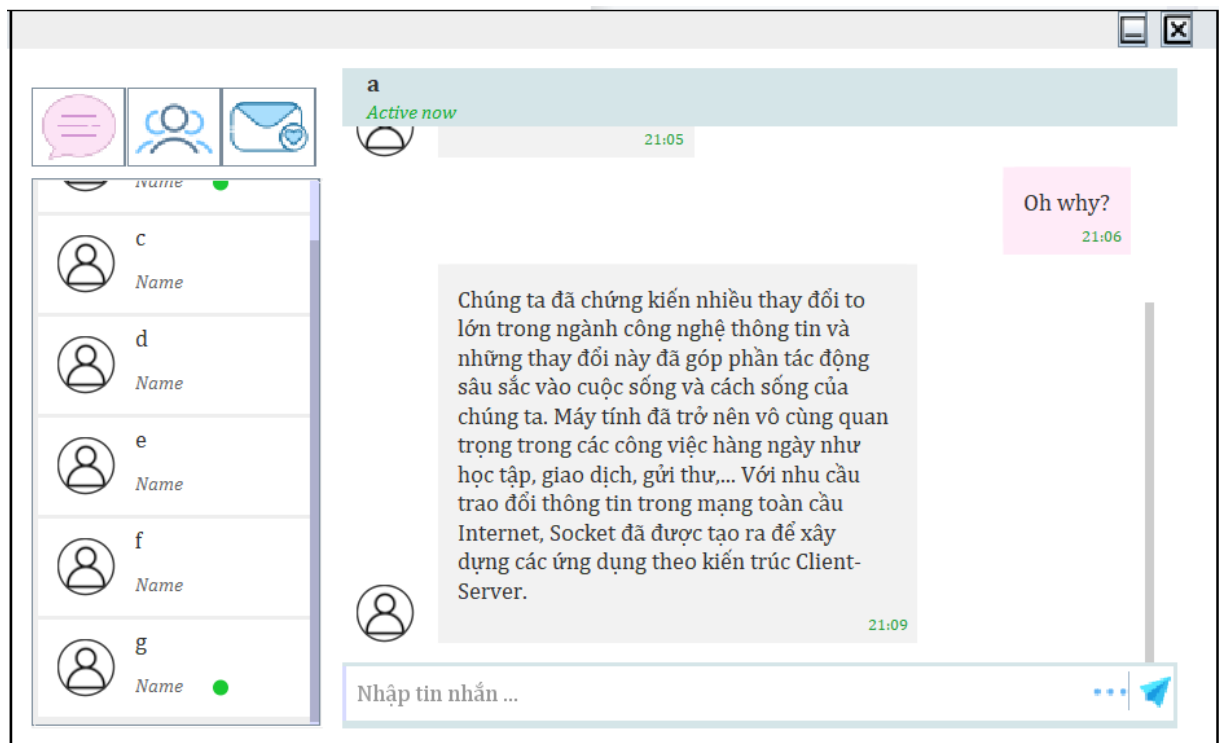
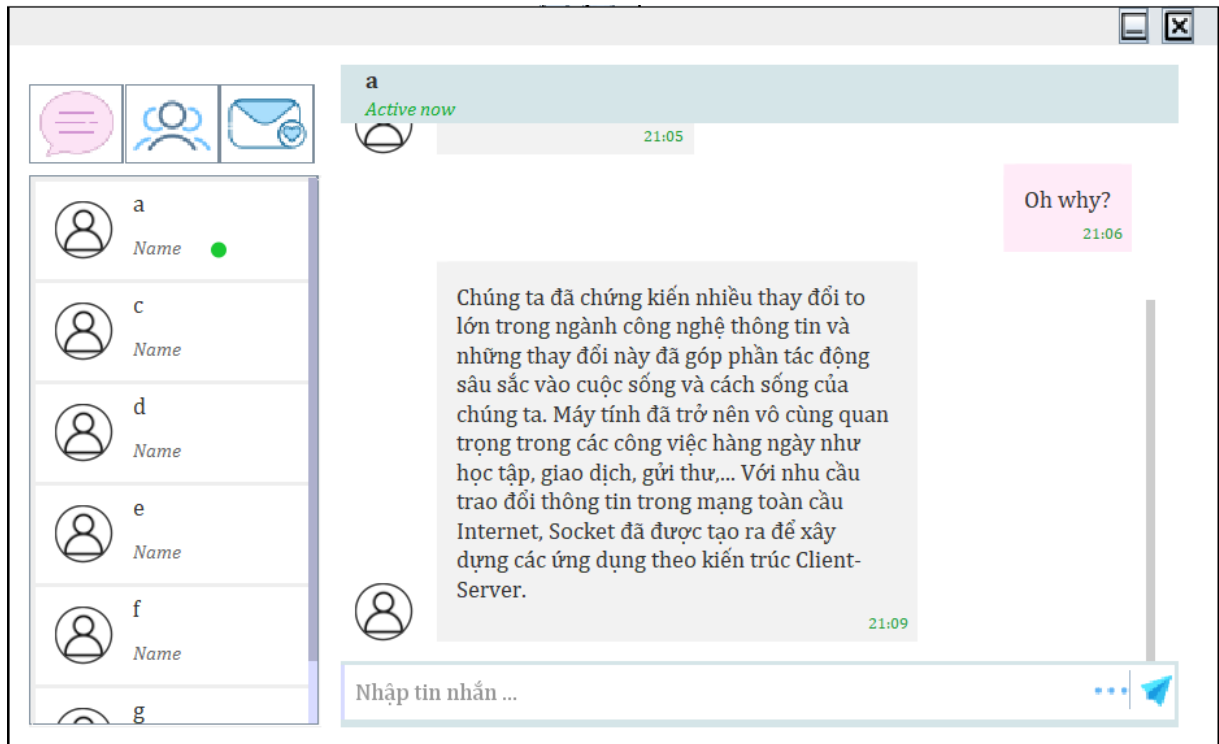
Hình 2.20 : Giao diện hệ thống Chat khi gõ một tin nhắn dài.



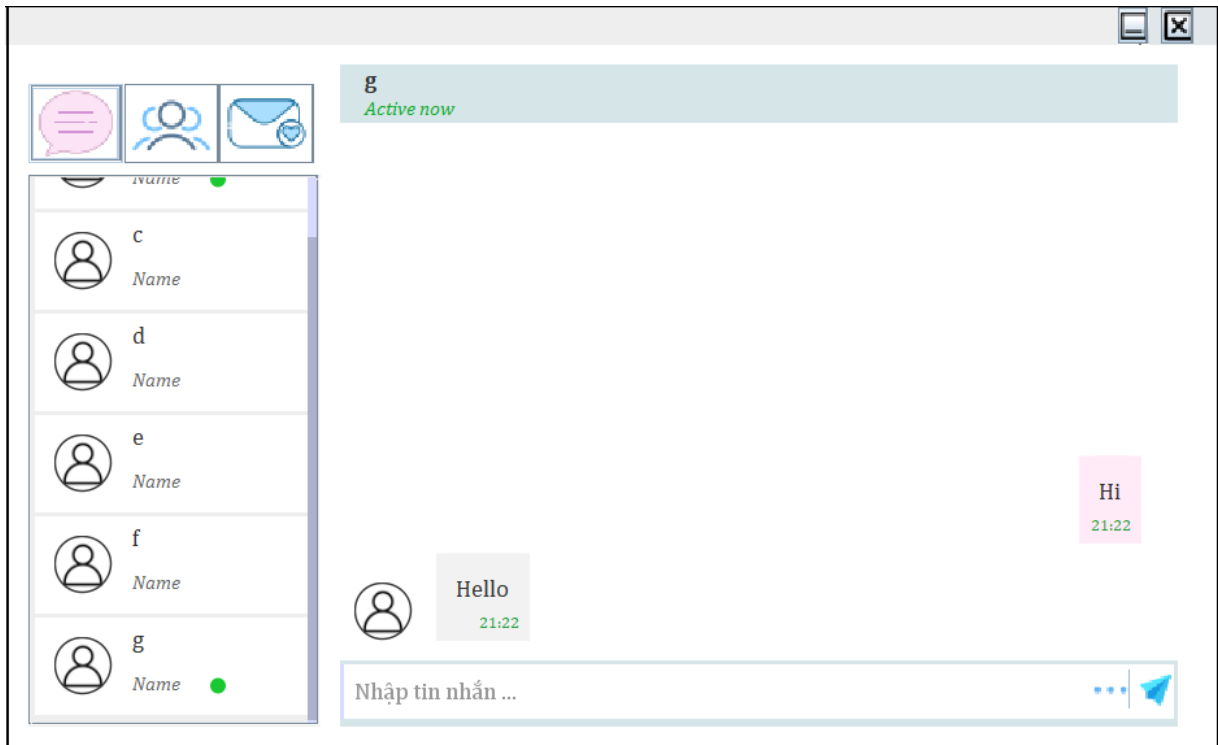
Hình 2.21 : Hệ thống Chat có thanh cuộn để người dùng có thể kéo lên và kéo xuống, dễ dàng cho việc đọc tin nhắn.

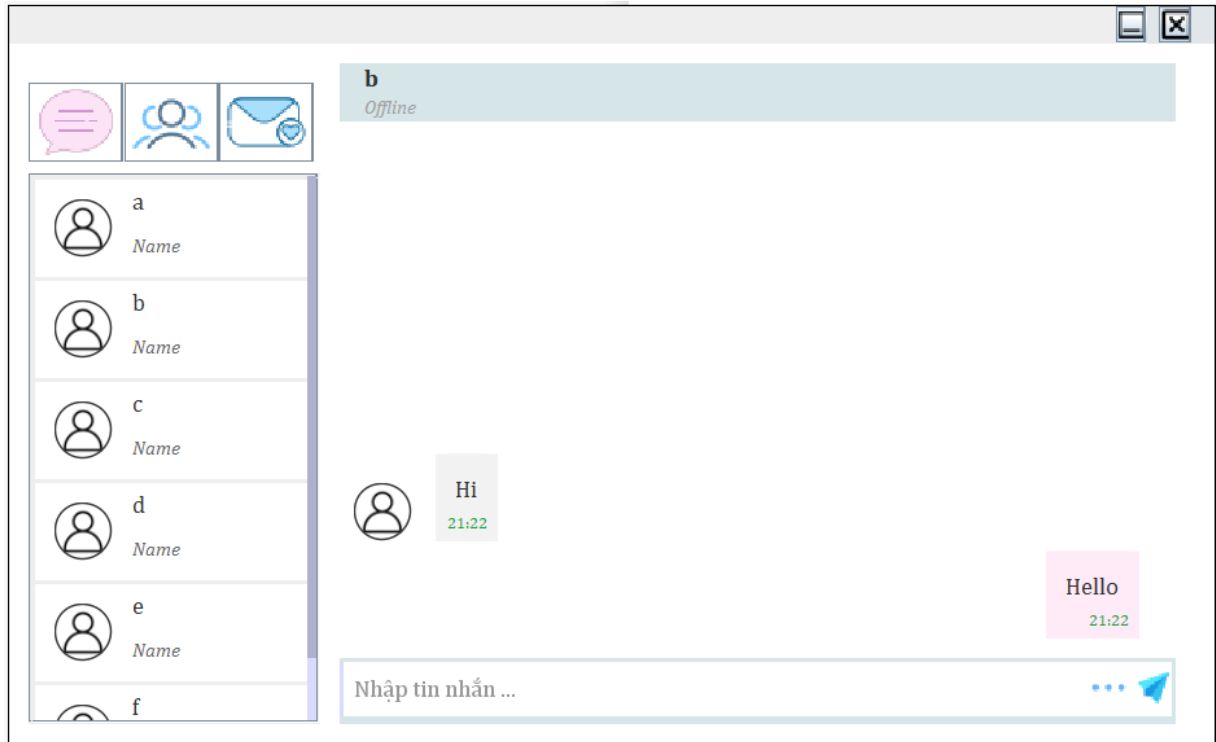


Hình 2.22 :Sau đó đăng ký thêm các tài khoản người dùng hệ thống Chat, và những thông tin này đều được lưu trên cơ sở dữ liệu itm_final.

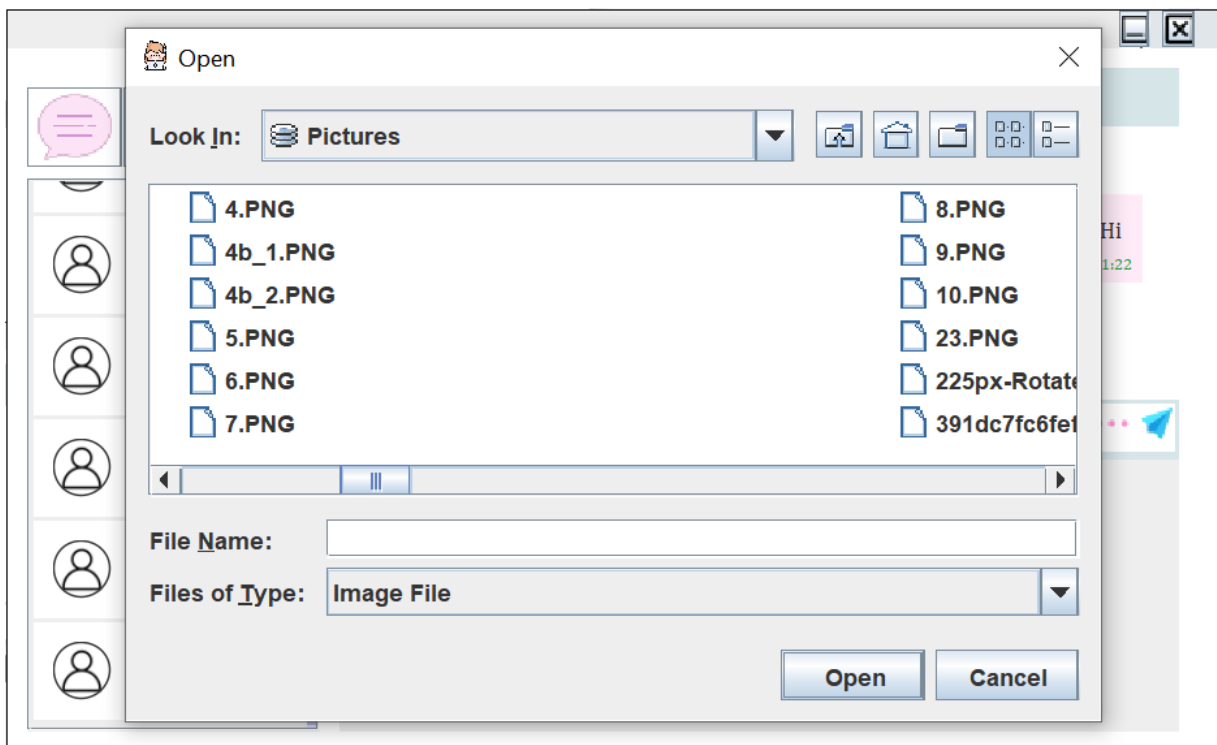
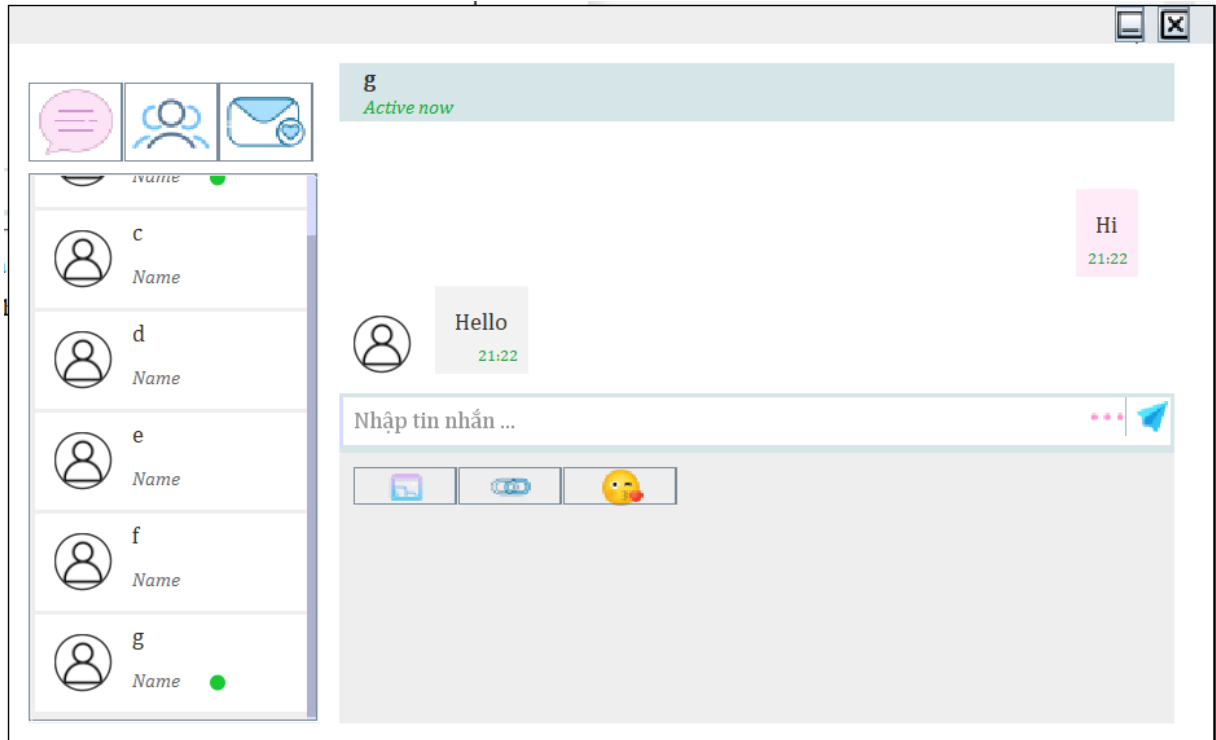


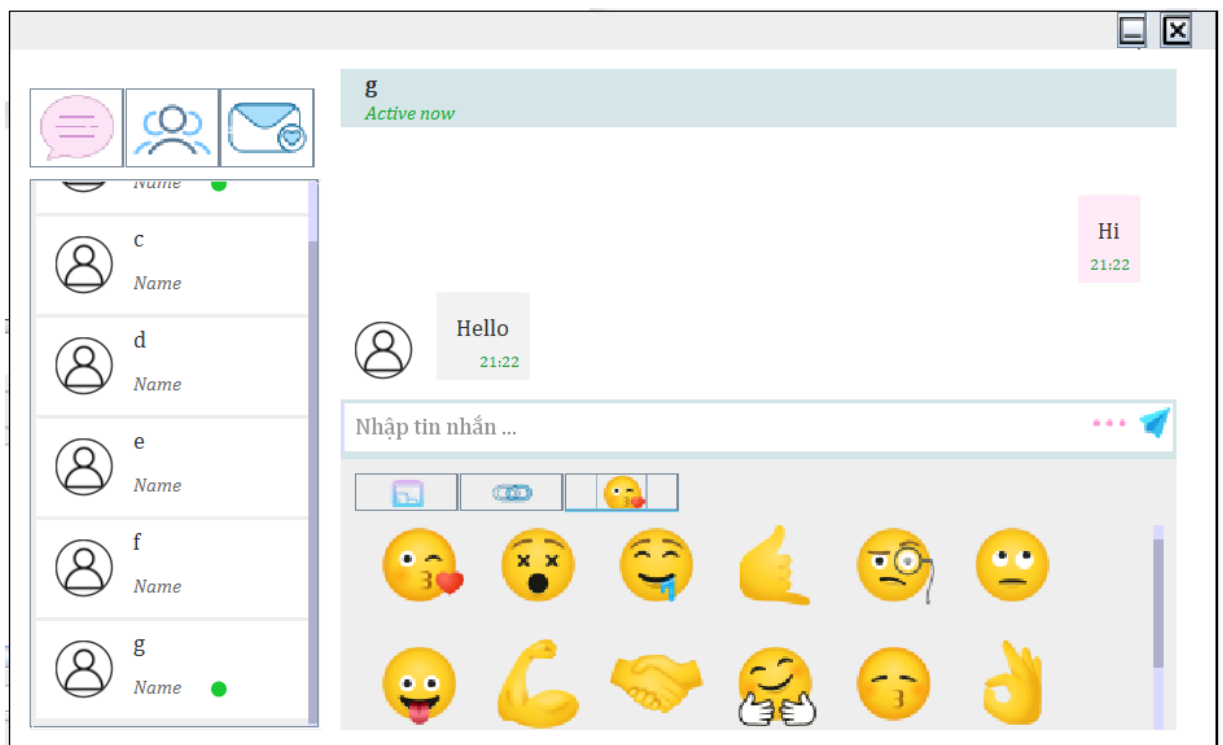
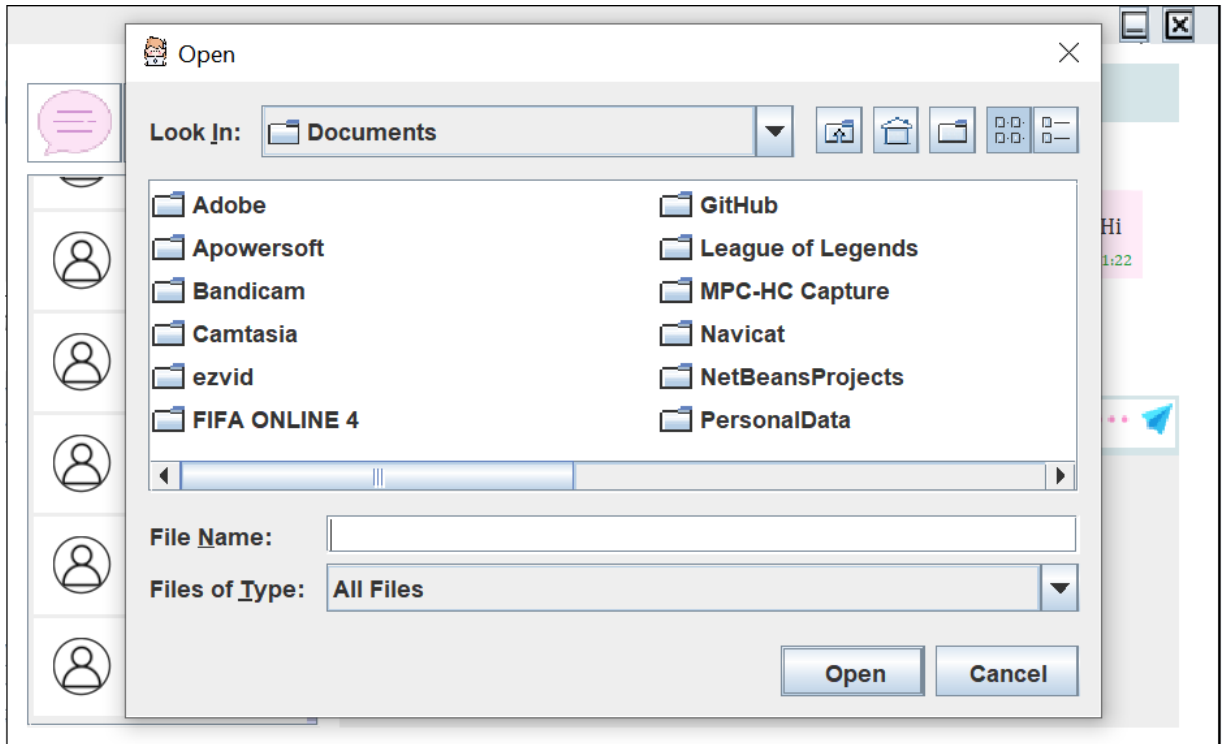
Giao diện Chat của người dùng “b” khi có thêm nhiều tài khoản khác đăng ký. Hệ thống sẽ có thanh cuộn để xem các thông tin các tài khoản này. Chấm tròn màu xanh có nghĩa là tài khoản đó hiện đang hoạt động.





Giao diện Chat của người dùng “b” và “g”. Hệ thống Chat sẽ xóa các đoạn tin nhắn trước đó nếu người dùng chọn một tài khoản khác để gửi tin nhắn.





Ngoài ra nhóm em có làm thêm các chức năng gửi hình ảnh, gửi file và gửi emoji, nhưng chưa thể hoàn thiện được.

Package Server:

File DatabaseConnection.java: package Server: Hàm public void connectToDatabase() throws SQLException sẽ thực hiện kết nối với cơ sở dữ liệu.

```
public void connectToDatabase() throws SQLException {
    String server = "localhost";
    Integer port = 3306;
    String dbname = "ltm_final";
    String username = "root";
    String password = "";

    MySQLDataSource datasource = new MySQLDataSource();
    datasource.setServerName(server);
    datasource.setUser(username);
    datasource.setPassword(password);
    datasource.setDatabaseName(dbname);
    datasource.setPortNumber(port);

    connection = datasource.getConnection();
}
```

File Service.java: Hàm public void startServer() thực hiện kết nối với Server.

```
public void startServer() {
    Configuration config = new Configuration();
    config.setPort(PORT_NUMBER);
    server = new SocketIOServer(config);

    server.addConnectListener(new ConnectListener() {
        @Override
        public void onConnect(SocketIOClient sioc) {
            textArea.append("Có một máy khách đã kết nối.\n");
        }
    });

    server.addEventListener("register", ModelRegister.class, new
    DataListener<ModelRegister>() {
        @Override
```

```

        public void onData(SocketIOClient sioc, ModelRegister t,
AckRequest ar) throws Exception {
            ModelMessage message = new ServiceUser().register(t);
            ar.sendAckData(message.isAction(),
message.getMessage(), message.getData());
            if (message.isAction()) {
                textArea.append("Người dùng đã đăng ký Username:"
+ t.getUsername() + ", Password:" + t.getPassword() + "\n");
                server.getBroadcastOperations().sendEvent("list_u
ser", (UserAccount) message.getData());
                addClient(sioc, (UserAccount) message.getData());
            }
        }
    });

    server.addEventListener("login", ModelLogin.class, new
DataListener<ModelLogin>() {
        @Override
        public void onData(SocketIOClient sioc, ModelLogin t,
AckRequest ar) throws Exception {
            UserAccount login = serviceUser.login(t);
            if (login != null) {
                ar.sendAckData(true, login);
                addClient(sioc, login);
                userConnect(login.getID());
            } else {
                ar.sendAckData(false);
            }
        }
    });

    server.addEventListener("list_user", Integer.class, new
DataListener<Integer>() {
        @Override
        public void onData(SocketIOClient sioc, Integer userID,
AckRequest ar) throws Exception {
            try {
                List<UserAccount> list =
serviceUser.getUser(userID);
                sioc.sendEvent("list_user", list.toArray());
            } catch (SQLException e) {
                System.err.println(e);
            }
        }
    });

```

```

    }
    });

    server.addListener("send_to_user", SendMessage.class,
new DataListener<SendMessage>() {
    @Override
    public void onData(SocketIOClient sioc, SendMessage t,
AckRequest ar) throws Exception {
        sendToClient(t);
    }
    });

    server.addDisconnectListener(new DisconnectListener() {
    @Override
    public void onDisconnect(SocketIOClient sioc) {
        int userID = removeClient(sioc);
        if (userID != 0) {
            // removed
            userDisconnect(userID);
        }
    }
    });

    server.start();
    textArea.append("Server đã khởi động với cổng " + PORT_NUMBER
+ ".\n");
}

```

File ServiceUser:

Khai báo câu lệnh truy vấn.

```

private final String INSERT_USER = "insert into `users`
(`username`, `password`) values (?,?)";
private final String INSERT_ACCOUNT = "insert into `accounts`
(`id`, `username`) values (?,?)";
private final String CHECK_USER = "SELECT * FROM `users` WHERE
`username` = ? limit 1";
private final String SELECT_ACCOUNT = "select * from `accounts`
where accounts.`status`='1' and id<>?";

```

```
private final String LOGIN = "select id, accounts.username,
gender, imageString from `users` join `accounts` using (id) where
`users`.username=BINARY(?) and `users`.`password`=BINARY(?) and
accounts.`Status`='1'";
```

Hàm public ModelMessage register(ModelRegister data) thực hiện chức năng đăng ký tài khoản cho người dùng.

```
public ModelMessage register(ModelRegister data) {
    // Check user exist
    ModelMessage message = new ModelMessage();
    try {
        PreparedStatement p = con.prepareStatement(CHECK_USER);
        p.setString(1, data.getUsername());
        ResultSet r = p.executeQuery();
        if (r.first()) {
            message.setAction(false);
            message.setMessage("Tên đăng nhập đã tồn tại!");
        } else {
            message.setAction(true);
        }
        r.close();
        p.close();
        if (message.isAction()) {
            // insert user vào database khi user đăng ký tài
khoản
            con.setAutoCommit(false);
            p = con.prepareStatement(INSERT_USER,
PreparedStatement.RETURN_GENERATED_KEYS);
            p.setString(1, data.getUsername());
            p.setString(2, data.getPassword());
            p.execute();
            r = p.getGeneratedKeys();
            r.first();
            int userID = r.getInt(1);
            r.close();
            p.close();

            //Tạo tài khoản cho user
            p = con.prepareStatement(INSERT_ACCOUNT);
            p.setInt(1, userID);
            p.setString(2, data.getUsername());
```

```

        p.execute();
        p.close();
        con.commit();
        con.setAutoCommit(true);
        message.setAction(true);
        message.setMessage("Ok");
        message.setData(new UserAccount(userID,
data.getUsername(), "", "", true));
    }
    } catch (SQLException e) {
        message.setAction(false);
        message.setMessage("Server Error");
        try {
            if (con.getAutoCommit() == false) {
                con.rollback();
                con.setAutoCommit(true);
            }
        } catch (SQLException e1) {
        }
    }
    return message;
}

```

Hàm public ModelMessage register(ModelRegister data) thực hiện chức năng đăng nhập tài khoản cho người dùng.

```

public UserAccount login(ModelLogin login) throws SQLException {
    UserAccount data = null;
    PreparedStatement p = con.prepareStatement(LOGIN);
    p.setString(1, login.getUserName());
    p.setString(2, login.getPassword());
    ResultSet r = p.executeQuery();
    if (r.first()) {
        int userID = r.getInt(1);
        String userName = r.getString(2);
        String gender = r.getString(3);
        String image = r.getString(4);
        data = new UserAccount(userID, userName, gender, image,
true);
    }
    r.close();
    p.close();
}

```

```

        return data;
    }

    public List<UserAccount> getUser(int exitUser) throws
SQLException {
        List<UserAccount> list = new ArrayList<>();
        PreparedStatement p = con.prepareStatement(SELECT_ACCOUNT);
        p.setInt(1, exitUser);
        ResultSet r = p.executeQuery();
        while (r.next()) {
            int userID = r.getInt(1);
            String userName = r.getString(2);
            String gender = r.getString(3);
            String image = r.getString(4);
            list.add(new UserAccount(userID, userName, gender, image,
true));
        }
        r.close();
        p.close();
        return list;
    }
}

```

Package Client:

File Service.java:

Hàm public void startServer():

```

public void startServer() {
    try {
        client = IO.socket("http://" + IP + ":" + PORT_NUMBER);
        client.on("list_user", new Emitter.Listener() {
            @Override
            public void call(Object... os) {
                // list user
                List<User_Account> users = new ArrayList<>();
                for (Object o : os) {
                    User_Account u = new User_Account(o);
                    if (u.getID() != user.getID()) {
                        users.add(u);
                    }
                }
            }
        });
    }
}

```

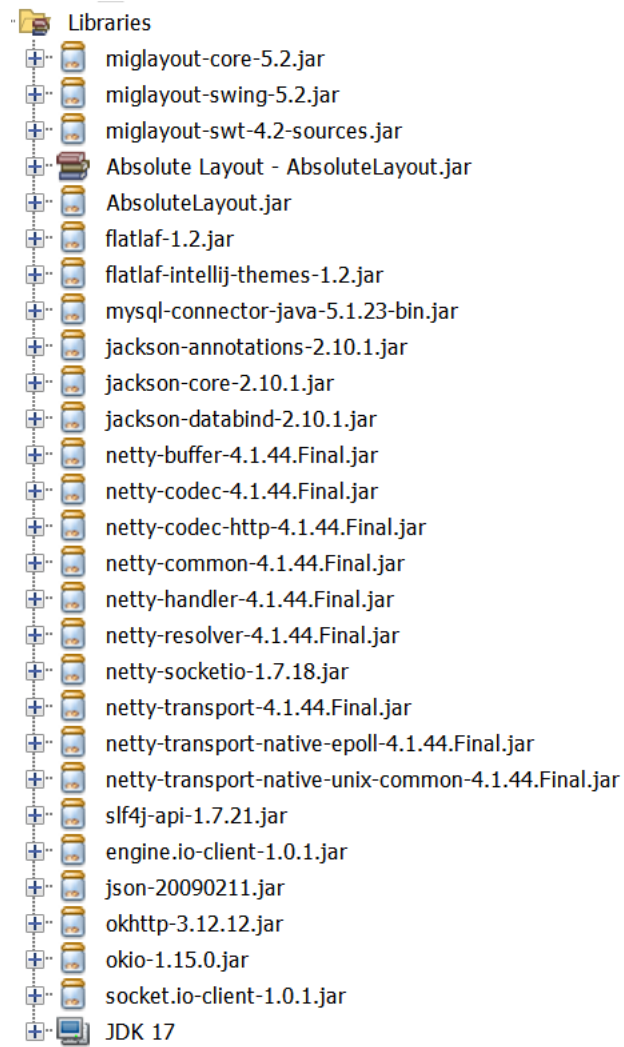
```

        PublicEvent.getInstance().getMenuLeftEvent().newU
ser(users);
    }
});

client.on("user_status", new Emitter.Listener() {
    @Override
    public void call(Object... os) {
        int userID = (Integer) os[0];
        boolean status = (Boolean) os[1];
        if (status) {
            // connect
            PublicEvent.getInstance().getMenuLeftEvent().
userConnect(userID);
        } else {
            // disconnect
            PublicEvent.getInstance().getMenuLeftEvent().
userDisconnect(userID);
        }
    }
});

client.on("receive_ms", new Emitter.Listener() {
    @Override
    public void call(Object... os) {
        Receive_Message message = new
Receive_Message(os[0]);
        PublicEvent.getInstance().getChatEvent().receiveM
sg(message);
    }
});
client.open();
} catch (URISyntaxException e) {
    error(e);
}
}

```

Hình 2.23 Các thư viện sử dụng trong hệ thống Chat.

CHƯƠNG 3 _ KẾT LUẬN

Nhìn chung, mô hình ứng dụng chat client server cũng đã hoàn thiện. Toàn bộ hệ thống đã được thiết kế trên java swing. Mô hình đáp ứng được các yêu cầu được đưa ra như các client và server có thể truyền thông điệp cho nhau và có các ứng dụng cơ bản .

TÀI LIỆU THAM KHẢO

1. <http://networkprogrammingnotes.blogspot.com/p/socket-addresses.html>
2. <http://etutorials.org/Programming/Pocket+pc+network+programming/Chapter+1.+Winsock/Socket+Options>
3. https://www.tutorialspoint.com/java/java_networking.htm