**PHẦN I: ĐẶT VẤN ĐỀ**

1. **Ý nghĩa, sự cần thiết và khả năng ứng dụng của đề tài**
2. **Ý nghĩa và sự cần thiết của phần mềm chat**

Từ xưa đến nay, giao tiếp là một trong những nhu cầu căn bản của con người. Giao tiếp giúp con người trao đổi thông tin với nhau điều khiển hành vi, phối hợp cũng như động viên khuyến khích nhau. Nhưng khi con người ở cách xa nhau, việc giao tiếp trở nên khó khăn hơn nhiều. Người ta phải sử dụng thư từ để trao đổi tin tức đi xa. Đến thế kỉ 20, khi công nghệ bán dẫn và linh kiện điện tử ra đời, việc giao tiếp từ xa đã trở nên dễ dàng hơn qua việc sử dụng mã Morse và sau đó là điện thoại. Bước vào kỉ nguyên số, việc trao đổi thông tin từ xa lại càng trở nên dễ dàng nhanh chóng và tiện lợi hơn trước nhờ có internet. Các phần mềm chat như Yahoo ra đời, tiếp sau đó là đến các mạng xã hội và các phần mềm chat hiện đại có thể tích hợp việc gọi điện có hình ảnh như Skype. Một số phần mềm còn được tích hợp riêng cho nền tảng di động như Zalo, Viber, ...

Như vậy có thể nói hiện nay, các phần mềm chat đang có vai trò quan trọng trong việc giao tiếp hiện nay. Hầu như mọi người dùng internet đều sử dụng ít nhất là 1 trong các phần mềm đó.

1. **Khả năng ứng dụng của đề tài**

Phần mềm chat trên mạng LAN của nhóm có thể được cài đặt để giao tiếp và chuyển file giữa các máy trong mạng nội bộ của một tổ chức, trường học, bệnh viện, công ty, ...

1. **Phân tích nội dung đề tài**

Phần mềm của nhóm được xây dựng bằng ngôn ngữ Java, một trong những ngôn ngữ lập trình phổ biến nhất thế giới với nhiều thư viện hỗ trợ. Phần mềm có những yêu cầu cơ bản sau:

* Yêu cầu chức năng:
* Có khả năng chat được giữa 2 người cũng như là có thể tạo nhóm chat để nhiều người chat cùng lúc với nhau được.
* Quản lí hoạt động chat bằng tài khoản (account) và mật khẩu (password), nhờ đó mà có thể lưu trữ lại lịch sử đoạn chat của người dùng
* Có thể gửi và nhận tệp tin chứa dữ liệu
* Có thể sử dụng các kí tự để tạo thành các biểu tượng cảm xúc (emoticon) trong đoạn chat
* Yêu cầu phi chức năng:
* Giao diện thân thiện, dễ sử dụng, quen thuộc với người dùng
* Hoạt động tương đối ổn định, ít xảy ra lỗi
* Sử dụng ít tài nguyên hệ thống
* Sử dụng hệ quản trị CSDL miễn phí: MySQL
* Không xung đột với các phần mềm khác, có khả năng nâng cấp, thêm các tính năng khác mà không ảnh hưởng đến CSDL
* Có khả năng mở rộng lên thành phần mềm chat trên mạng WAN (internet)

**PHẦN II: PHƯƠNG PHÁP GIẢI QUYẾT**

Dựa vào những yêu cầu đặt ra ở trên, nhóm đã nghiên cứu và thống nhất phương án xây dựng phần mềm như sau:

1. **Mô hình hoạt động của hệ thống**

****

*Hình 1: Mô hình hệ thống*

Hệ thống được thiết kế theo mô hình client – server với client là các máy dùng phần mềm chat trong mạng, server là máy chủ trong mạng. Trên máy server chứa cơ sở dữ liệu của toàn bộ hệ thống. Các client không trực tiếp tương tác với nhau mà bất kì tương tác dữ liệu nào đều phải thông qua server. Server sẽ xử lí thông tin mà các client gửi đến, xử lí dữ liệu đó trên CSDL và sau đó sẽ trả về client thông tin cần tương ứng.

Tất cả mọi quá trình truyền tải thông tin giữa server và client có thể được tóm tắt như sau:

* B1: Client tạo 1 luồng thiết lập kết nối với server và client
* B2: Client gửi thông tin mà người dùng tương tác trên giao diện đến server đi kèm theo đó là 1 mã lệnh (command) để server nhận biết là client cần thực hiện điều gì
* B3: Server nhận biết mã lệnh, thực hiện chức năng tương ứng bằng việc xử lí cơ sở dữ liệu hoặc tập tin trên server, sau đó trả thông tin về cho client (vẫn trong kết nối đã lập)
* B4: Client nhận được thông tin, xử lí thông tin đã nhận được trên giao diện, đóng kết nối lại
* B5: Server nhận được thông báo rằng kết nối từ client đã đóng thì sẽ đóng kết nối phía server lại và kết thúc luồng

1. **Phương thức truyền tải thông tin**

Trong 2 phương thức TCP và UDP, nhóm quyết định lựa chọn phương thức TCP để truyền thông tin trên mạng LAN giữa server và client do những lí do sau:

* Việc truyền bằng TCP giúp dữ liệu chính xác, đảm bảo trật tự của gói tin. Điều này là rất quan trọng đối với tính năng gửi file.
* Tốc độ của TCP tuy chậm hơn tuy nhiên do phần mềm được thiết kế để gửi những gói tin nhỏ nên nhược điểm này của TCP có thể chấp nhận được.
* Giao thức TCP có khả năng quản lí và điều khiển luồng tốt hơn so với UDP, nhờ đó tránh được việc thiết kế không tốt dẫn đến tổn hao nhiều tài nguyên hệ thống.

1. **Đối tượng truyền tải thông tin**

Do thông tin truyền tải giữa client và server cần phải có nhiều kiểu dữ liệu chứ không chỉ là chuỗi văn bản, nên cần có 1 đối tượng truyền tải thông tin thống nhất trong hệ thống.

Nhóm đã xây dựng đối tượng có tên là DataSend để giải quyết vấn đề trên với các thuộc tính như sau:

* command: chứa câu lệnh. Thuộc tính này giúp cho server nhận biết được cần thực hiện lệnh gì
* yourUserName: chứa username của người gửi tin
* password/time: chứa password (khi thực hiện lệnh đăng nhập/đăng kí) hoặc chứa thời gian của dòng chat
* yourFriendUserName: chứa username của người nhận tin
* message: chưa nội dung tin nhắn
* fileName: chứa tên tập tin (chức năng gửi file)
* data: chứa nội dung tập tin (chức năng gửi file)

1. **Ý tưởng về thuật toán chương trình**

* Giao diện của phần mềm client lấy ý tưởng từ giao diện trang <http://www.messenger.com>, bao gồm 2 phần chính:
* Phần bên trái là danh sách các cuộc trò chuyện gần đây. Danh sách này sẽ được xây dựng bằng Java Swing JList.
* Phần bên phải có 1 JTextPane chứa cuộc trò chuyện hiện tại đang sử dụng. Ở dưới có JTextField để người dùng nhập đoạn chat vào và 2 nút Send và Attach File cho tính năng gửi file
* Sẽ có 1 luồng trên giao diện chính liên tục gửi tin đến server đòi truy vấn CSDL mỗi 2s để refresh lại danh sách các cuộc trò chuyện. Mỗi khi người dùng bấm vào 1 phần tử của JList, sẽ có 1 thuộc tính của MainForm lưu lại xem người dùng đã bấm vào tên ai
* Có 1 luồng khác trên giao diện sẽ liên tục cập nhật cứ mỗi 1s JTextPane chứa nội dung chat giữa username hiện tại với username mà người dùng vừa bấm vào ở JList bằng việc gửi tin đến server để truy vấn CSDL. Bằng cách lưu lại lịch sử chat trên CSDL server nên đảm bảo mỗi lần khởi động chương trình, tin nhắn sẽ không mất đi và có thể xem lại được
* Tính năng gửi/nhận file: Client sẽ chuyển nội dung file cần gửi thành 1 mảng byte, sau đó gửi tới server thông qua đối tượng truyền tin. Server sẽ lưu lại file với tên vào ổ cứng. Lịch sử chat cũng sẽ lưu lại thông tin này trên CSDL và tạo ra trên JTextPane một JButton có nhãn là tên file cần gửi. Khi người dùng bấm vào JButton đó thì client sẽ tạo ra kết nối yêu cầu server gửi file đó cho mình bằng cách tương tự khi gửi.
* Tính năng gửi tin nhắn có emoticon: khi truy vấn tin nhắn từ CSDL về client, client sẽ xử lí xâu tin nhắn đó, tìm ra xem chỗ nào có chuỗi kí tự đặc biệt như :D, :v, :x, ... thì sẽ thay thế chúng bằng emoticon gần tương ứng như trong Facebook
* Tính năng thay avatar:
* Client khi refresh danh sách chat gần đây sẽ tìm trên ổ cứng client xem có file avatar của username đó không. Nếu không có thì sẽ yêu cầu server gửi file avatar của username đó. Nếu trên server cũng không có thì client sẽ sử dụng avatar mặc định.
* Khi bấm vào nút đổi avatar của mình trên JMenuBar của MainForm, người dùng sẽ chọn file avatar từ cửa sổ JFileChooser. Sau đó client sẽ gửi file avatar đó đến server.

**PHẦN III: PHÂN TÍCH THIẾT KẾ PHẦN MỀM**

1. **Cấu trúc các phần của phần mềm**
2. **Cơ sở dữ liệu**

* Bảng tbl\_user: chứa thông tin các tài khoản

|  |  |  |  |
| --- | --- | --- | --- |
| **Tên** | **Mô tả** | **Kiểu dữ liệu** | **Khác** |
| userName | Tên tài khoản | varchar(16) | Khóa chính, Unique, Not Null |
| password | Mật khẩu | varchar(20) | not null |

* Bảng tbl\_conversation: chứa thông tin về các tin nhắn trò chuyện

|  |  |  |  |
| --- | --- | --- | --- |
| **Tên** | **Mô tả** | **Kiểu dữ liệu** | **Khác** |
| id | STT tin nhắn | int | Khóa chính, Auto increasement, unique, not null |
| userSend | Tài khoản gửi | varchar(16) | not null |
| userReceive | Tài khoản nhận tin nhắn | varchar(16) | not null |
| message | Tin nhắn trò chuyện | varchar |  |
| time | Thời gian gửi tin nhắn | datetime | not null |

1. **Server**

Bao gồm các class:

* ***DataSend implements Serializable:*** đối tượng truyền tin
* ***MultiThreadTCPEchoServer:***lớp chứa phương thức main của Server, khởi tạo ra 1 luồng đối với mỗi đối tượng ServerSocket nhận được
* ***TCPEchoThread implements Runnable:*** lớp kế thừa từ Runnable triển khai chạy đa luồng
* *Contructor:* thực hiện việc phân loại lệnh dựa vào mã lệnh của đối tượng DataSend, từ đó mà thực hiện việc truy vấn cở sở dữ liệu hoặc thực thi với file trên đĩa
* *private void writeFileToDisk (DataSend dataReceive):* thực hiện việc ghi tập tin vào ổ cứng server (chức năng gửi file)
* *private byte[] readFileFromDisk (DataSend dataReceive):* thực hiện việc đọc tập tin ra từ ổ cứng server, đưa nội dung tập tin vào 1 mảng byte
* *private void writeAvatarToDisk(DataSend dataReceive):* thực hiện việc ghi lại tập tin chứa avatar của username có thông tin chứa trong *dataReceive* (chức năng đổi avatar)
* *private byte[] readAvatarFromDisk(DataSend dataReceive):*
* ***QuerryData:*** lớp thực hiện truy vấn với CSDL
* *public void QuerrySignIn(DataSend dataReceive):* truy vấn thông tin cho chức năng đăng nhập
* *public void QuerrySignUp(DataSend dataReceive):* truy vấn thông tin cho chức năng đăng ký
* *public void AddUser(DataSend dataReceive):* thêm mới username vào CSDL nếu như đăng kí thành công
* *public void QuerryListChat (DataSend dataReceive):* truy vấn danh sách những cuộc trò chuyện gần đây của 1 username
* *public void QuerryChatWindow (DataSend dataReceive):* truy vấn cuộc trò chuyện của 1 username với 1 username khác / nhóm chat
* *public void AddMessage(DataSend dataReceive):* thêm mới 1 tin nhắn trò chuyện
* *public void SendFile (DataSend dataReceive):* thêm mới thông tin về việc có file được gửi
* *public void AddConversation(DataSend dataReceive):* thêm mới 1 cuộc trò chuyện
* *public void QuerryUser (DataSend dataReceive):* kiểm tra xem username cần gửi tin nhắn đến ở chức năng thêm mới cuộc trò chuyện có nằm trong CSDL hay không

1. **Client**

Được thiết kế theo mô hình MVC:

* ***Model:***
  + ***DataSend implements Serializable:*** đối tượng truyền tin
* ***Conversation implements IConversation****:* Đối tượng cuộc trò chuyện hiển thị trong Jlist
* *Contructor*: thực hiện khởi tạo các thuộc tính của đối tượng displayName (tên hiển thị), name (tên tài khoản), mess (tin nhắn gần nhất), date (thời gian gửi tính từ 01/01/1970 tính ra mili giây).
* Các method *getDisplayName(), getDate(), getMess(), getName()* lấy các giá trị tên hiển thị, ngày, tin nhắn gần nhất và tên
* Method *getMinAgo()* lấy ra giá trị tin nhắn gần nhất được gửi cách thời điểm hiện tại bao nhiêu phút
* ***ConvRenderer extends JPanel implements ListCellRenderer<Conversation>****:* thực hiện thay thế Jlist với mỗi element là 1 đối tượng bằng 1 Panel thể hiện cuộc trò chuyện
* *Contructor*: Thực hiện tạo panel và add các component bao gồm: 1 label icon, 1 label tên hiển thị, 1 label tin nhắn gần nhất và 1 label thời gian gửi tin nhắn gần nhất
* *getListCellRendererComponent(JList<? extends Conversation> list, Conversation convers, int index, boolean isSelected, boolean cellHasFocus)*: được override lại method trong *ListCellRenderer* trả về component đã được cấu hình để hiển thị các giá trị trong từng label đã tạo ở contructor.
* ***View:***
* ***MainView extends Jframe implements IMainView****:* hiển thị giao diện cửa sổ chính của phần mềm
* *Contructor*: Thực hiện khởi tạo các component và add các component vào Frame bao gồm:
* 1 menu bar có 1 menu với 3 item: new message, change avatar và log out
* 1 list thể hiện các cuộc trò chuyện với hiển thị gồm 1 avatar, tên người dùng, tin nhắn gần nhất và thời gian gửi tin nhắn gần nhất
* 1 TextPane hiển thị phần tin nhắn chat
* 1 TextField để nhập nội dung chat
* 1 button gửi tin nhắn và 1 button để đính kèm file
* Các method tạo sự kiện cho các button, menu item và Jlist
* *public void addActionListenerForChangeAvatarInMainForm(ActionListener listener)*: tạo sự kiện thay đổi avatar
* *public void addActionListenerForSignOutInMainForm(ActionListener listener)*: tạo sự kiện sign out
* *public void addActionListenerForNewMessageInMainForm(ActionListener listener)*: tạo sự kiện tạo tin nhắn mới
* *public void addActionListenerForAttachFileInMainForm(ActionListener listener):* tạo sự kiện đính kèm file
* *public void addActionListenerForSendButtonInMainForm(ActionListener listener):* tạo sự kiện gửi tin nhắn
* *addActionListenerForJListInMainForm(ListSelectionListener listener):* tạo sự kiện lấy element từ Jlist khi click chuột chọn
* *public void insertStringInPane(int size, String text, Color cl)*: thực hiện chèn String vào TextPane để hiển thị tên người dùng với kích thước và màu sắc khác
* *public String getTextToSend()*: thực hiện lấy String từ TextField và set TextField = “”
* *public void handleString(String string)* xử lý xâu thay thế 1 số các emoticon
* *public void handleReceiver(String string)*: xử lý xâu nhận về từ server xem đó là dòng tin nhắn hay là file đính kèm để xử lý, nếu là file thực hiện tạo Button cho phép người dùng download còn nếu là tin nhắn bình thường thì xử lý thay thế imoticon
* *public void resetList(Conversation[] conv)* : cập nhật List
* *public String getSelectJList()* lấy tên của phần tử được chọn trong Jlist
* *public void setScrollBottom()*: Set thanh cuộn của TextPane xuống dưới cùng để người dùng nhìn thấy tin nhắn gần nhất khi lịch sử tin nhắn dài
* *public void clearJTextPane()*: xóa dữ liệu trên TextPane
* ***NewMessageForm extends JFrame implements InewMessageForm:*** hiển thị giao diện thêm mới cuộc trò chuyện
* *Contructor:* Thực hiện khởi tạo các component và add các component vào Frame bao gồm:
* JLabel và JTextField chứa tên những username cần thêm cuộc trò chuyện mới, các username ngăn cách nhau bằng dấu cách (space)
* Nút OK và nút Cancel
* *public void closeForm():* Đóng form giao diện lại
* *public void setActionListenerForOKButtonInNewMassageForm(ActionListener listener):* tạo ActionListener cho sự kiện bấm bút OK
* *public String getTextUserName():* phương thức lấy ra nội dung của JTextField chứa tên các username
* ***SignInForm extends JFrame implements IsignInForm:*** giao diện đăng nhập vào phần mềm (cũng là giao diện khởi động phần mềm)
* *Contructor:* Thực hiện khởi tạo các component và add các component vào Frame bao gồm:
* JLabel và JTextField chứa username
* JLabel và JPasswordField chứa password
* Nút Sign In, Register, Exit
* *public void closeForm():* Đóng form và kết thúc chương trình
* *public void closeWhenSignInToMainView():* đóng form khi đăng nhập thành công để vào giao diện chính
* *public void setActionListenerForSignInButtonInSignInForm(ActionListener listener):* tạo ActionListener cho sự kiện bấm nút SignIn
* *public void setActionListenerForRegisterButtonInSignInForm(ActionListener listener):* tạo ActionListener cho sự kiện bấm nút Register
* *public void setActionListenerForExitButtonInSignInForm(ActionListener listener):* tạo ActionListener cho sự kiện bấm nút Exit
* *public String getTextUserName():* trả về nội dung của JTextField username
* *public String getTextPassword():* trả về nội dung của JPassword password
* ***SignUpForm extends JFrame implements IsignUpForm:*** giao diện đăng kí tài khoản để sử dụng phần mềm
* *Contructor: :* Thực hiện khởi tạo các component và add các component vào Frame bao gồm:
* JLabel và JTextField chứa username
* JLabel và JPasswordField chứa password
* JLabel và JPasswordField bắt người dùng nhập lại lần nữa password
* Nút Sign Up, Exit
* *public void closeForm():* Đóng form
* *public void setActionListenerForSignUpButtonInSignUpForm(ActionListener listener) :* tạo ActionListener cho sự kiện bấm nút SignUp
* ***Controller:***
* ***Chat:*** Chứa phương thức main để chạy chương trình Client, mở ra 1 SignInController mới
* ***MainFormController:*** điều khiển hoạt động trên giao diện chính
  + *Contructor:* 
    - Thiết đặt ActionListener khi bấm vào các nút và JMenuBar trên giao diện
    - Khởi tạo và chạy luồng cập nhật các cuộc trò chuyện
    - Khởi tạo và chạy luồng cập nhật cửa sổ chat hiện tại
* *private void updateListChat():* Phương thức thực thi thuật toán cho luồng cập nhật cuộc trò chuyện
* *private void updateChatWindow():* Phương thức thực thi thuật toán cho luồng cập nhật cửa sổ chat
* *public static void receiveFile (String fileName):* Phương thức thực hiện việc nhận file khi bấm vào nút có chứa tên file trong cửa sổ chat
* ***NewMessageController:*** điều khiển hoạt động trên giao diện thêm tin nhắn mới
* ***SignInController:*** điều khiển hoạt động trên giao diện đăng nhập
* ***SignUpController:*** điều khiển hoạt động trên giao diện đăng kí tài khoản mới

1. **Mô tả hoạt động của hệ thống**
2. **Đăng nhập**



*Hình 2: Sơ đồ quá trình đăng nhập*

* Người dùng nhập username và password vào. Khi bấm vào nút Sign In, client sẽ tạo kết nối gửi đến server đối tượng DataSend có command = signIn.
* Server sẽ truy vấn CSDL xem có username đó không và trả về cho client cũng qua đối tượng DataSend

1. **Đăng kí**

* Tương tự như quá trình đăng kí, nhưng trước hết client sẽ kiểm tra xem password và re-enter password có trùng nhau không

1. **Gửi tin nhắn**



*Hình 3: Sơ đồ quá trình gửi tin nhắn*

1. **Thêm mới cuộc trò chuyện**



*Hình 4: Sơ đồ quá trình thêm mới cuộc trò chuyện*

1. **Cập nhật cửa sổ trò chuyện**



*Hình 5: Sơ đồ quá trình cập nhật cửa sổ trò chuyện*

1. **Cập nhật cuộc trò truyện gần đây**



*Hình 6: Sơ đồ quá trình cập nhật các cuộc trò chuyện gần đây*

1. **Gửi file**



*Hình 7 : Sơ đồ quá trình gửi file*

1. **Nhận file**



*Hình 8: Sơ đồ quá trình nhận file*

1. **Thay đổi avatar**



*Hình 9: Sơ đồ quá trình thay đổi avatar*

**PHẦN IV: CODE CHƯƠNG TRÌNH**

Code của 1 số class quan trọng trong chương trình

1. **Server. TCPEchoThread**
2. **package** Server;
3. **import** java.io.\*;
4. **import** java.net.\*;
5. **import** java.sql.SQLException;
6. **import** java.nio.file.Files;
7. **import** java.nio.file.Paths;
8. **import** java.nio.file.Path;
9. **public** **class** TCPEchoThread **implements** Runnable {
10. **private** Socket socket;
11. **public** TCPEchoThread(Socket s){
12. socket = s;
13. }
14. **public** **void** run() {
15. **try**(ObjectInputStream in = **new** ObjectInputStream(socket.getInputStream());
16. ObjectOutputStream out = **new** ObjectOutputStream(socket.getOutputStream())
17. ){
18. DataSend dataReceive = (DataSend) in.readObject();
19. DataSend dataSend = **new** DataSend();
20. System.***out***.println("DataReceive: "+dataReceive.getInfo());
21. QuerryData querryData = **new** QuerryData();
22. String command = dataReceive.getInfo().split("\\|")[0];
23. System.***out*** .println("Command: "+command);
24. //command = "signIn";
25. **int** commandID = 0;
26. //do something here
27. **if** (command.equals("signIn"))
28. commandID = 1;
29. **else** **if**(command.equals("signUp"))
30. commandID = 2;
31. **else** **if**(command.equals("sendMsg"))
32. commandID = 3;
33. **else** **if**(command.equals("updateListChat"))
34. commandID = 4;
35. **else** **if**(command.equals("updateChatWindow"))
36. commandID = 5;
37. **else** **if**(command.equals("sendFile"))
38. commandID = 6;
39. **else** **if**(command.equals("receiveFile"))
40. commandID = 7;
41. **else** **if**(command.equals("addConversation"))
42. commandID = 8;
43. **else** **if**(command.equals("changeAvatar"))
44. commandID = 9;
45. **else** **if**(command.equals("updateAvatar"))
46. commandID = 10;
47. **switch** (commandID){
48. **case** 1:
49. querryData.QuerrySignIn(dataReceive);
50. dataSend = querryData.data;
51. System.***out***.println("DataSend: "+dataSend.getInfo());
52. out.writeObject(dataSend);
53. out.flush();
54. **break**;
55. ////////////////////////////////
56. **case** 2:
57. querryData.QuerrySignUp(dataReceive);
58. dataSend = querryData.data;
59. **if** (querryData.data.getInfo().split("\\|")[0].equals("SignUpOK"))
60. querryData.AddUser(dataReceive);
61. out.writeObject(dataSend);
62. System.***out***.println("DataSend: "+dataSend.getInfo());
63. out.flush();
64. **break**;
65. ////////////////////////////////
66. **case** 3:
67. querryData.AddMessage(dataReceive);
68. **break**;
69. ///////////////////////////////
70. **case** 4:
71. querryData.QuerryListChat(dataReceive);
72. dataSend = querryData.data;
73. System.***out***.println("DataSend: "+dataSend.getInfo());
74. out.writeObject(dataSend);
75. out.flush();
76. **break**;
77. //////////////////////////////
78. **case** 5:
79. querryData.QuerryChatWindow(dataReceive);
80. dataSend = querryData.data;
81. //querryData.SetSeen(dataReceive);
82. out.writeObject(dataSend);
83. out.flush();
84. **break**;
85. //////////////////////////////
86. **case** 6:
87. querryData.SendFile(dataReceive);
88. writeFileToDisk(dataReceive);
89. **break**;
90. /////////////////////////////
91. **case** 7:
92. dataSend = **new** DataSend ("receiveFile", **null**, **null**,**null**,**null**, dataReceive.getFileName(), readFileFromDisk(dataReceive));
93. out.writeObject(dataSend);
94. out.flush();
95. **break**;
96. ////////////////////////////
97. **case** 8 :
98. querryData.QuerryUser(dataReceive);
99. **if**(querryData.data.getInfo().split("\\|")[0].equals("AddConversationOK"))
100. querryData.AddConversation(dataReceive);
101. dataSend = querryData.data;
102. System.***out***.println(dataSend.getInfo());
103. out.writeObject(dataSend);
104. out.flush();
105. **break**;
106. ///////////////////////////
107. **case** 9:
108. writeAvatarToDisk(dataReceive);
109. **break**;
110. //////////////////////////
111. **case** 10:
112. File fileAva = **new** File("D:\\Samsung\\Chat\\Server\\Avatar\\"+dataReceive.getYourUserName());
113. **if**(fileAva.exists()){
114. dataSend = **new** DataSend ("updateAvatarOK", **null**, **null**, **null**, **null**, **null**, readAvatarFromDisk(dataReceive));
115. }
116. **else**{
117. dataSend = **new** DataSend ("updateAvatarNotOK", **null**, **null**, **null**, **null**, **null**,**null**);
118. }
119. out.writeObject(dataSend);
120. out.flush();
121. **break**;
122. //////////////////////////
123. **default**:
124. **break**;
125. }
126. in.close();
127. out.close();
128. socket.close();
129. }**catch** (IOException e){
130. e.printStackTrace();
131. } **catch** (ClassNotFoundException e) {
132. // **TODO** Auto-generated catch block
133. System.***out***.println(e.getMessage());
134. } **catch** (SQLException e) {
135. // **TODO** Auto-generated catch block
136. e.printStackTrace();
137. }
138. }
140. **private** **void** writeFileToDisk (DataSend dataReceive){
141. String fileName = dataReceive.getFileName();
142. **try**{
143. FileOutputStream fos = **new** FileOutputStream("D:\\Samsung\\Chat\\Server\\File\\"+fileName);
144. fos.write(dataReceive.getData());
145. fos.close();
146. }
147. **catch** (FileNotFoundException e) {
148. // **TODO** Auto-generated catch block
149. e.printStackTrace();
150. } **catch** (IOException e) {
151. // **TODO** Auto-generated catch block
152. e.printStackTrace();
153. }
154. }
156. **private** **byte**[] readFileFromDisk (DataSend dataReceive){
157. **byte** [] data = **null**;
158. **try**{
159. String fileName = dataReceive.getFileName();
160. Path path = Paths.*get*("D:\\Samsung\\Chat\\Server\\File\\"+fileName);
161. data = Files.*readAllBytes*(path);
162. }
163. **catch** (IOException e) {
164. // **TODO** Auto-generated catch block
165. e.printStackTrace();
166. }
167. **return** data;
168. }
170. **private** **void** writeAvatarToDisk(DataSend dataReceive){
171. String fileName = dataReceive.getFileName();
172. **try**{
173. Path path = Paths.*get*("D:\\Samsung\\Chat\\Server\\Avatar\\"+fileName);
174. **if**(**new** File("D:\\Samsung\\Chat\\Server\\Avatar\\"+fileName).exists())
175. Files.*delete*(path);
176. FileOutputStream fos = **new** FileOutputStream("D:\\Samsung\\Chat\\Server\\Avatar\\"+fileName);
177. fos.write(dataReceive.getData());
178. fos.close();
179. }
180. **catch** (FileNotFoundException e) {
181. // **TODO** Auto-generated catch block
182. e.printStackTrace();
183. } **catch** (IOException e) {
184. // **TODO** Auto-generated catch block
185. e.printStackTrace();
186. }
187. }
189. **private** **byte**[] readAvatarFromDisk(DataSend dataReceive){
190. **byte** [] data = **null**;
191. **try**{
192. String fileName = dataReceive.getYourUserName();
193. Path path = Paths.*get*("D:\\Samsung\\Chat\\Server\\Avatar\\"+fileName);
194. data = Files.*readAllBytes*(path);
195. }
196. **catch** (IOException e) {
197. // **TODO** Auto-generated catch block
198. e.printStackTrace();
199. }
200. **return** data;
201. }
203. }

**2. Server.QuerryData**

1. **package** Server;
2. **import** java.sql.Connection;
3. **import** java.sql.DriverManager;
4. **import** java.sql.PreparedStatement;
5. **import** java.sql.ResultSet;
6. **import** java.sql.SQLException;
7. **import** java.sql.Statement;
8. **import** java.sql.Timestamp;
9. **import** java.text.DateFormat;
10. **import** java.text.SimpleDateFormat;
11. **import** java.util.Date;
12. **public** **class** QuerryData {
13. **public** DataSend data = **new** DataSend();
14. **public** QuerryData() {
15. // **TODO** Auto-generated constructor stub
16. }
18. **public** QuerryData(DataSend dataReceive) {
19. String command = dataReceive.getCommand();
20. }
22. **public** **void** QuerrySignIn(DataSend dataReceive) **throws** SQLException{
23. String userName = dataReceive.getInfo().split("\\|")[1];
24. String password = dataReceive.getInfo().split("\\|")[2];
25. Connection conn;
26. Statement stmt = **null**;
27. conn = DriverManager.*getConnection*("jdbc:mysql://localhost:6688/chat", "root", "");
28. stmt = conn.createStatement();
29. String queryStr = "SELECT \* FROM tbl\_user WHERE (userName = '"+userName+"');";
30. ResultSet rs = stmt.executeQuery(queryStr);
31. **if**(!rs.first())
32. data.setCommand("SignInNotOK");
33. **else**{
34. **if** (rs.getString("password").equals(password))
35. data.setCommand("SignInOK");
36. **else**
37. data.setCommand("SignInNotOK");
38. }
39. }
41. //////////////////////
43. **public** **void** QuerrySignUp(DataSend dataReceive) **throws** SQLException{
44. String userName = dataReceive.getInfo().split("\\|")[1];
45. //String password = dataReceive.getInfo().split("|")[2];
46. Connection conn;
47. Statement stmt = **null**;
48. conn = DriverManager.*getConnection*("jdbc:mysql://localhost:6688/chat", "root", "");
49. stmt = conn.createStatement();
50. String queryStr = "SELECT \* FROM tbl\_user WHERE (userName = '"+userName+"');";
51. ResultSet rs = stmt.executeQuery(queryStr);
52. **if**(!rs.first())
53. data.setCommand("SignUpOK");
54. **else**
55. data.setCommand("SignUpNotOK");
56. }
58. ///////////////////////////////
60. **public** **void** AddUser(DataSend dataReceive) **throws** SQLException{
61. String userName = dataReceive.getInfo().split("\\|")[1];
62. String password = dataReceive.getInfo().split("\\|")[2];
63. Connection conn;
64. Statement stmt = **null**;
65. conn = DriverManager.*getConnection*("jdbc:mysql://localhost:6688/chat", "root", "");
66. stmt = conn.createStatement();
67. String queryStr = "INSERT IGNORE INTO tbl\_user VALUES(?,?);";
68. PreparedStatement addStmt = conn.prepareStatement(queryStr);
69. addStmt.setString(1, userName);
70. addStmt.setString(2, password);
71. addStmt.addBatch();
72. addStmt.executeBatch();
73. }
75. ///////////////////////////////
76. **public** **void** QuerryListChat (DataSend dataReceive)**throws** SQLException{
77. String yourUser = dataReceive.getInfo().split("\\|")[1];
78. String yourFriendUserList = "";
79. String messageList = "";
80. String timeList = "";
81. Connection conn;
82. Statement stmt = **null**;
83. conn = DriverManager.*getConnection*("jdbc:mysql://localhost:6688/chat", "root", "");
84. stmt = conn.createStatement();
85. String queryStr = "SELECT userSend, userReceive, message, time FROM (SELECT \* from tbl\_conversation ORDER BY time DESC limit 50)t WHERE (userSend = '"+yourUser+"') OR (userReceive LIKE '% "+yourUser+" %') GROUP BY userReceive ORDER BY time DESC;";
86. ResultSet rs = stmt.executeQuery(queryStr);
87. **if**(!rs.first())System.***out***.println("Have no record!");
88. **else**{
89. //display result if not empty
90. **do**{
91. String yourFriendUser = rs.getString(2).replaceAll("\\|", "").trim();
92. String message = rs.getString(3).replaceAll("\\|", "").trim();
93. Timestamp dt = rs.getTimestamp(4);
94. Long time = dt.getTime();
95. yourFriendUserList = yourFriendUserList+yourFriendUser+"\_";
96. messageList = messageList + message + "\_";
97. timeList = timeList+ time.toString() + "\_";
98. }**while**(rs.next());
99. }
100. data = **new** DataSend("updateListChat", **null**, timeList, yourFriendUserList, messageList, **null**, **null**);
101. }
102. ///////////////////////////
104. **public** **void** QuerryChatWindow (DataSend dataReceive) **throws** SQLException{
105. String yourUser = dataReceive.getInfo().split("\\|")[1];
106. String yourFriendUser = dataReceive.getYourFriendUserName();
107. Connection conn;
108. String messageList = "";
109. String userSendList = "";
110. Statement stmt = **null**;
111. conn = DriverManager.*getConnection*("jdbc:mysql://localhost:6688/chat", "root", "");
112. stmt = conn.createStatement();
113. String queryStr = "SELECT message, time, userSend, userReceive, seen FROM tbl\_conversation WHERE (userReceive = '"+yourFriendUser+"');";
114. ResultSet rs = stmt.executeQuery(queryStr);
115. **if**(!rs.first())
116. data = **new** DataSend("updateChatWindow", " ", " ", " ", " ", " ", **new** **byte**[2]);
117. **else**{
118. //display result if not empty
119. **do**{
120. String message = rs.getString(1).replaceAll("\\|", "").trim();
121. String userSend = rs.getString(3).replaceAll("\\|", "").trim();
122. messageList = messageList + message + "\_";
123. userSendList = userSendList + userSend + "\_";
124. } **while**(rs.next());
125. }
126. data = **new** DataSend("updateChatWindow", yourUser, userSendList, yourFriendUser, messageList, **null**, **null**);
127. }
129. /////////////////////////////
131. **public** **void** AddMessage(DataSend dataReceive) **throws** SQLException{
132. String yourUser = dataReceive.getInfo().split("\\|")[1];
133. String yourFriendUser = dataReceive.getYourFriendUserName();
134. String message = dataReceive.getMessage();
135. Connection conn;
136. Statement stmt = **null**;
137. conn = DriverManager.*getConnection*("jdbc:mysql://localhost:6688/chat", "root", "");
138. stmt = conn.createStatement();
139. String queryStr = "INSERT IGNORE INTO tbl\_conversation VALUES(NULL,?,?,?,?,?);";
140. PreparedStatement addStmt = conn.prepareStatement(queryStr);
141. DateFormat dateFormat = **new** SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
142. Date date = **new** Date();
143. String time = dateFormat.format(date);
144. addStmt.setString(1, yourUser);
145. addStmt.setString(2, yourFriendUser);
146. addStmt.setString(3, message);
147. addStmt.setString(4, time);
148. addStmt.setInt(5, 0);
149. addStmt.addBatch();
150. addStmt.executeBatch();
151. }
153. **public** **void** SendFile (DataSend dataReceive) **throws** SQLException{
154. String yourUser = dataReceive.getYourUserName();
155. String yourFriendUser = dataReceive.getYourFriendUserName();
156. String fileName = dataReceive.getFileName();
157. Connection conn;
158. Statement stmt = **null**;
159. conn = DriverManager.*getConnection*("jdbc:mysql://localhost:6688/chat", "root", "");
160. stmt = conn.createStatement();
161. String queryStr = "INSERT IGNORE INTO tbl\_conversation VALUES(NULL,?,?,?,?,?);";
162. PreparedStatement addStmt = conn.prepareStatement(queryStr);
163. DateFormat dateFormat = **new** SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
164. Date date = **new** Date();
165. String time = dateFormat.format(date);
166. addStmt.setString(1, yourUser);
167. addStmt.setString(2, yourFriendUser);
168. addStmt.setString(3, "<file>"+fileName);
169. addStmt.setString(4, time);
170. addStmt.setInt(5, 0);
171. addStmt.addBatch();
172. addStmt.executeBatch();
173. }
175. **public** **void** AddConversation(DataSend dataReceive) **throws** SQLException {
176. String yourUser = dataReceive.getYourUserName();
177. String yourFriendUser = dataReceive.getYourFriendUserName();
178. Connection conn;
179. Statement stmt = **null**;
180. conn = DriverManager.*getConnection*("jdbc:mysql://localhost:6688/chat", "root", "");
181. stmt = conn.createStatement();
182. String queryStr = "INSERT IGNORE INTO tbl\_conversation VALUES(NULL,?,?,?,?,?);";
183. PreparedStatement addStmt = conn.prepareStatement(queryStr);
184. DateFormat dateFormat = **new** SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
185. Date date = **new** Date();
186. String time = dateFormat.format(date);
187. addStmt.setString(1, yourUser);
188. addStmt.setString(2, yourFriendUser);
189. addStmt.setString(3, "|null|");
190. addStmt.setString(4, time);
191. addStmt.setInt(5, 0);
192. addStmt.addBatch();
193. addStmt.executeBatch();
194. }
196. **public** **void** QuerryUser (DataSend dataReceive) **throws** SQLException {
197. String yourFriendUser = dataReceive.getYourFriendUserName();
198. Connection conn;
199. Statement stmt = **null**;
200. conn = DriverManager.*getConnection*("jdbc:mysql://localhost:6688/chat", "root", "");
201. stmt = conn.createStatement();
202. **int** i = 1;
203. **boolean** check = **true**;
204. **try**{
205. **while** (yourFriendUser.split("\\s+")[i] != **null**){
206. String queryStr = "SELECT \* from tbl\_user WHERE (userName = '"+yourFriendUser.split("\\s+")[i]+"');";
207. ResultSet rs = stmt.executeQuery(queryStr);
208. **if**(!rs.first()){
209. check = **false**;
210. **break**;
211. }
212. i++;
213. }
214. }
215. **catch** (ArrayIndexOutOfBoundsException e ) {
216. System.***out***.println("Split done");
217. }
218. **if** (check)
219. data.setCommand("AddConversationOK");
220. **else** data.setCommand("AddConversationNotOK");
221. }
223. **public** **void** SetSeen(DataSend dataReceive) **throws** SQLException{
224. String yourFriendUser = dataReceive.getYourFriendUserName();
225. String queryStr = "UPDATE tbl\_conversation SET seen = 1 WHERE (userReceive = '"+yourFriendUser+"') AND (message<>'|');";
226. Connection conn;
227. Statement stmt = **null**;
228. conn = DriverManager.*getConnection*("jdbc:mysql://localhost:6688/chat", "root", "");
229. stmt = conn.createStatement();
230. stmt.executeUpdate(queryStr);
231. }
233. }
234. **DataSend**
235. **package** Server;
236. **import** java.io.Serializable;
237. **public** **class** DataSend **implements** Serializable{
238. **private** String command;
239. **private** String yourUserName;
240. **private** String password;
241. **private** String yourFriendUserName;
242. **private** String message;
243. **private** String fileName;
244. **private** **byte**[] data;
246. **public** DataSend() {
247. // **TODO** Auto-generated constructor stub
249. }
251. **public** DataSend(String command, String username, String password, String userSend, String message, String fileName, **byte**[] data){
252. **this**.command = command;
253. **this**.yourUserName = username;
254. **this**.password = password;
255. **this**.yourFriendUserName = userSend;
256. **this**.message = message;
257. **this**.fileName= fileName;
258. **this**.data = data;
259. }
261. **public** **void** setCommand(String command){
262. **this**.command = command;
263. }
265. **public** String getInfo(){
266. **return** command+"|"+yourUserName+"|"+password+"|"+yourFriendUserName+"|"+message+"|"+fileName;
267. }
269. **public** String getYourUserName(){
270. **return** **this**.yourUserName;
271. }
273. **public** String getYourFriendUserName(){
274. **return** **this**.yourFriendUserName;
275. }
277. **public** String getTimelist(){
278. **return** **this**.password;
279. }
281. **public** String getMessage(){
282. **return** **this**.message;
283. }
285. **public** String getFileName(){
286. **return** **this**.fileName;
287. }
289. **public** **byte**[] getData(){
290. **return** data;
291. }
293. **public** String getCommand(){
294. **return** command;
295. }
296. }
297. **Client.model.ConvRenderer**
298. **package** Client.model;
299. **import** java.awt.BorderLayout;
300. **import** java.awt.Color;
301. **import** java.awt.Component;
302. **import** java.awt.GridLayout;
303. **import** java.io.File;
304. **import** javax.swing.ImageIcon;
305. **import** javax.swing.JLabel;
306. **import** javax.swing.JList;
307. **import** javax.swing.JPanel;
308. **import** javax.swing.ListCellRenderer;
309. **public** **class** ConvRenderer **extends** JPanel **implements** ListCellRenderer<Conversation> {
310. **private** JLabel lbIcon = **new** JLabel();
311. **private** JLabel lbName = **new** JLabel();
312. **private** JLabel lbMess = **new** JLabel();
313. **private** JLabel lbTime = **new** JLabel();
315. **public** ConvRenderer(){
316. setLayout(**new** BorderLayout(5, 5));
318. JPanel panelText = **new** JPanel(**new** GridLayout(0, 1));
319. panelText.add(lbName);
320. panelText.add(lbMess);
321. panelText.add(lbTime);
322. add(lbIcon, BorderLayout.***WEST***);
323. add(panelText, BorderLayout.***CENTER***);
324. }
325. @Override
326. **public** Component getListCellRendererComponent(JList<? **extends** Conversation> list, Conversation convers, **int** index,
327. **boolean** isSelected, **boolean** cellHasFocus) {
328. // **TODO** Auto-generated method stub
329. File fileAva = **new** File("src/avatar/"+convers.getDisplayName());
330. **if** (fileAva.exists())
331. lbIcon.setIcon(**new** ImageIcon(**new** ImageIcon("src/avatar/"+convers.getDisplayName()).getImage().getScaledInstance(40, 40, java.awt.Image.***SCALE\_SMOOTH***)));
332. **else**
333. lbIcon.setIcon(**new** ImageIcon(**new** ImageIcon("src/avatar/default.jpg").getImage().getScaledInstance(40, 40, java.awt.Image.***SCALE\_SMOOTH***)));
334. lbName.setText(convers.getDisplayName());
335. lbMess.setText(convers.getMess());
336. **long** min = convers.getMinAgo();
337. **if** (min < 60)
338. lbTime.setText(Long.*toString*(min)+ ((min <= 1)?" minute ago":" minutes ago"));
339. **else**{
340. **long** hour = min/60;
341. **long** min1 = min%60;
342. **if** (hour < 24){
343. lbTime.setText(Long.*toString*(hour)+ ((hour == 1)?" hour ":" hours ")
344. + ((min1 == 0)?"":(Long.*toString*(min1)+((min1 <= 1)?" minute":" minutes")))+" ago");
345. }
346. **else**{
347. lbTime.setText(convers.getDate());
348. }
349. }
350. lbName.setForeground(Color.***red***);
351. lbTime.setForeground(Color.***blue***);

354. // set Opaque to change background color of JLabel
355. lbName.setOpaque(**true**);
356. lbMess.setOpaque(**true**);
357. lbTime.setOpaque(**true**);
358. lbIcon.setOpaque(**true**);
360. // when select item
361. **if** (isSelected) {
362. lbName.setBackground(list.getSelectionBackground());
363. lbMess.setBackground(list.getSelectionBackground());
364. lbTime.setBackground(list.getSelectionBackground());
365. lbIcon.setBackground(list.getSelectionBackground());
366. setBackground(list.getSelectionBackground());
367. } **else** { // when don't select
368. lbName.setBackground(list.getBackground());
369. lbMess.setBackground(list.getBackground());
370. lbTime.setBackground(list.getBackground());
371. lbIcon.setBackground(list.getBackground());
372. setBackground(list.getBackground());
373. }
374. **return** **this**;
375. }
376. }
377. **Client.view.MainForm**
378. **package** Client.view;
379. **import** java.awt.BorderLayout;
380. **import** java.awt.Color;
381. **import** java.awt.Component;
382. **import** java.awt.GridLayout;
383. **import** java.awt.Point;
384. **import** java.awt.event.ActionEvent;
385. **import** java.awt.event.ActionListener;
386. **import** java.awt.event.KeyEvent;
387. **import** java.awt.event.KeyListener;
388. **import** java.util.Date;
389. **import** javax.swing.BorderFactory;
390. **import** javax.swing.DefaultListModel;
391. **import** javax.swing.ImageIcon;
392. **import** javax.swing.JButton;
393. **import** javax.swing.JFrame;
394. **import** javax.swing.JList;
395. **import** javax.swing.JMenu;
396. **import** javax.swing.JMenuBar;
397. **import** javax.swing.JMenuItem;
398. **import** javax.swing.JPanel;
399. **import** javax.swing.JScrollPane;
400. **import** javax.swing.JTextField;
401. **import** javax.swing.JTextPane;
402. **import** javax.swing.SwingUtilities;
403. **import** javax.swing.event.DocumentEvent;
404. **import** javax.swing.event.DocumentListener;
405. **import** javax.swing.event.ListSelectionListener;
406. **import** javax.swing.text.AbstractDocument;
407. **import** javax.swing.text.BadLocationException;
408. **import** javax.swing.text.SimpleAttributeSet;
409. **import** javax.swing.text.Style;
410. **import** javax.swing.text.StyleConstants;
411. **import** javax.swing.text.StyledDocument;
412. **import** javax.swing.text.Utilities;
413. **import** Client.controller.MainFormController;
414. **import** Client.model.ConvRenderer;
415. **import** Client.model.Conversation;
416. **public** **class** MainForm **extends** JFrame **implements** IMainForm {
418. //emoticon
419. ImageIcon angry = **new** ImageIcon("src/icon/angry.png");
420. **final** String strAngry = ">:O";
421. ImageIcon big\_smile = **new** ImageIcon("src/icon/big\_smile.png");
422. **final** String strBigSmile = ":D";
423. ImageIcon cry = **new** ImageIcon("src/icon/cry.png");
424. **final** String strCry = ":'(";
425. ImageIcon curly\_lips = **new** ImageIcon("src/icon/curly\_lips.png");
426. **final** String strCurlyLips = ":3";
427. ImageIcon glasses = **new** ImageIcon("src/icon/glasses.png");
428. **final** String strGlasses = "8-)";
429. ImageIcon heart = **new** ImageIcon("src/icon/heart.png");
430. **final** String strHeart = "<3";
431. ImageIcon kiki = **new** ImageIcon("src/icon/kiki.png");
432. **final** String strKiki = "^\_^";
433. ImageIcon kiss = **new** ImageIcon("src/icon/kiss.png");
434. **final** String strKiss = ":\*";
435. ImageIcon like = **new** ImageIcon("src/icon/like.png");
436. **final** String strLike = "(y)";
437. ImageIcon pacman = **new** ImageIcon("src/icon/pacman.png");
438. **final** String strPacman = ":v";
439. ImageIcon sad = **new** ImageIcon("src/icon/sad.png");
440. **final** String strSad = ":(";
441. ImageIcon squinting = **new** ImageIcon("src/icon/squinting.png");
442. **final** String strsquinting = "-\_-";
443. ImageIcon tongue\_out = **new** ImageIcon("src/icon/tongue\_out.png");
444. **final** String strTongueOut = ":P";
445. ImageIcon unsure = **new** ImageIcon("src/icon/unsure.png");
446. **final** String strUnsure = ":/";
447. ImageIcon wink = **new** ImageIcon("src/icon/wink.png");
448. **final** String strWink = ";)";
449. ImageIcon wtf = **new** ImageIcon("src/icon/wtf.png");
450. **final** String strWtf = "o.O";
452. String[] emoticons = {strAngry, strBigSmile, strCry, strCurlyLips, strGlasses,
453. strHeart, strKiki, strKiss, strLike, strPacman, strSad, strsquinting,
454. strTongueOut, strUnsure, strWink, strWtf};
456. **public** JList myList;
457. **private** JMenuItem btnAdd = **new** JMenuItem("New",KeyEvent.***VK\_N***);
458. **private** JMenuItem btnOut = **new** JMenuItem("Sign out",KeyEvent.***VK\_G***);
459. **private** JMenuItem btnChangeAvatar = **new** JMenuItem ("Change Avatar", KeyEvent.***VK\_C***);
460. **private** JButton btnSend = **new** JButton("Send");
461. **private** JButton btnAtt = **new** JButton("Attach File");
462. **private** JTextField tfMess = **new** JTextField(20);
463. **private** JTextPane taShow = **new** JTextPane();
464. **private** JButton btnTemp = **new** JButton(" ");
465. **private** JScrollPane scrollShow;
467. **private** JList<Conversation> createList() {
468. // create defaultListModel
469. DefaultListModel<Conversation> model = **new** DefaultListModel<>();
470. // add element to model
472. // set model to JList
473. JList<Conversation> list = **new** JList<Conversation>(model);
474. list.setCellRenderer(**new** ConvRenderer());
475. **return** list;
476. }


480. **public** MainForm(String userName){
482. //display Menu bar
483. JMenuBar menuBar = **new** JMenuBar();
484. JMenu menu = **new** JMenu("Option");
485. menu.setMnemonic(KeyEvent.***VK\_O***);
486. menuBar.add(menu);
487. menu.add(btnAdd);
488. menu.add(btnChangeAvatar);
489. menu.add(btnOut);
490. setJMenuBar(menuBar);
492. //display list
493. myList = createList(); //create list
494. JScrollPane scrollList = **new** JScrollPane(myList);
495. scrollList.setBorder(BorderFactory.*createEmptyBorder*(
496. 5, //top
497. 5, //left
498. 5, //bottom
499. 0) //right
500. );
502. taShow.setEditable(**false**);

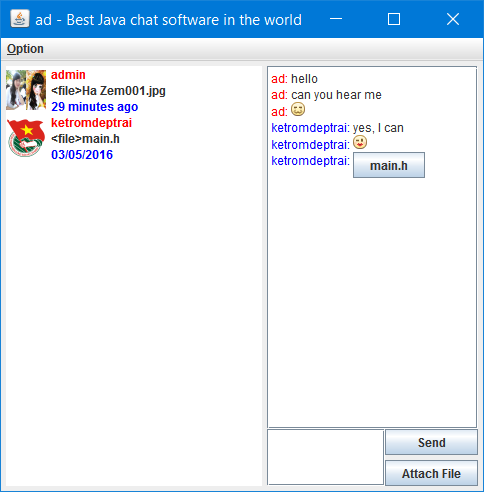
505. //panel send message (right)
506. JPanel pnRight = **new** JPanel(**new** BorderLayout());
508. //panel includes text field, 2 buttons are send and attach
509. JPanel pnUnder = **new** JPanel(**new** BorderLayout());
511. //panel Send: 2 buttons are send and attach
512. JPanel pnSend = **new** JPanel(**new** GridLayout(0,1,5,5));
513. pnSend.add(btnSend);
514. pnSend.add(btnAtt);
515. pnUnder.add(pnSend,BorderLayout.***EAST***);
516. btnSend.requestFocus();
517. pnUnder.add(tfMess,BorderLayout.***CENTER***);
518. // enter for tfMess
519. tfMess.addKeyListener(**new** KeyListener(){
520. @Override
521. **public** **void** keyPressed(KeyEvent arg0) {
522. // **TODO** Auto-generated method stub
523. **if** (arg0.getKeyCode() == KeyEvent.***VK\_ENTER***)
524. btnSend.doClick();
525. }
526. @Override
527. **public** **void** keyReleased(KeyEvent arg0) {
528. // **TODO** Auto-generated method stub
529. }
530. @Override
531. **public** **void** keyTyped(KeyEvent arg0) {
532. // **TODO** Auto-generated method stub
533. }
534. });
535. //Text Panes
536. scrollShow = **new** JScrollPane(taShow);
537. taShow.setVisible(**true**);
539. //Show
540. pnRight.add(scrollShow,BorderLayout.***CENTER***);
541. pnRight.add(pnUnder,BorderLayout.***SOUTH***);
542. pnRight.setBorder(BorderFactory.*createEmptyBorder*(
543. 5, //top
544. 5, //left
545. 5, //bottom
546. 5) //right
547. );
549. // add panel to container
550. setLayout(**new** BorderLayout());
551. add(scrollList,BorderLayout.***WEST***);
552. add(pnRight,BorderLayout.***CENTER***);
554. setTitle(userName + " - Best Java chat software in the world");
555. setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);
556. setSize(500,500);
557. setLocationRelativeTo(**null**);
558. setVisible(**true**);
559. }
560. **public** **void** addActionListenerForTemp(ActionListener listener){
561. **this**.btnTemp.addActionListener(listener);
562. }
563. @Override
564. **public** **void** addActionListenerForChangeAvatarInMainForm(ActionListener listener){
565. **this**.btnChangeAvatar.addActionListener(listener);
566. }
568. **public** **void** addActionListenerForSignOutInMainForm(ActionListener listener) {
569. // **TODO** Auto-generated method stub
570. **this**.btnOut.addActionListener(listener);
571. }
572. @Override
573. **public** **void** addActionListenerForNewMessageInMainForm(ActionListener listener) {
574. // **TODO** Auto-generated method stub
575. **this**.btnAdd.addActionListener(listener);
576. }
577. @Override
578. **public** **void** addActionListenerForAttachFileInMainForm(ActionListener listener) {
579. // **TODO** Auto-generated method stub
580. **this**.btnAtt.addActionListener(listener);
581. }
582. @Override
583. **public** **void** addActionListenerForSendButtonInMainForm(ActionListener listener) {
584. // **TODO** Auto-generated method stub
585. **this**.btnSend.addActionListener(listener);
586. }
587. @Override
588. **public** **void** closeForm() {
589. // **TODO** Auto-generated method stub
590. **this**.dispose();
591. }
593. @Override
594. **public** **void** inSert(Component component) {
595. // **TODO** Auto-generated method stub
596. taShow.insertComponent(component);
597. }
598. @Override
599. **public** **void** insertStringInPane(**int** size, String text, Color cl) {
600. // **TODO** Auto-generated method stub
601. StyledDocument doc = (StyledDocument) taShow.getDocument();
602. // Create a style object and then set the style attributes
603. Style style = doc.addStyle("Sty", **null**);
604. StyleConstants.*setFontSize*(style, size);
605. StyleConstants.*setForeground*(style,cl);
606. **try** {
607. doc.insertString(doc.getLength(), text, style);
608. } **catch** (BadLocationException e) {
609. // **TODO** Auto-generated catch block
610. e.printStackTrace();
611. }
613. }
614. @Override
615. **public** String getTextToSend(){
616. String text = **this**.tfMess.getText() + '\n';
617. **this**.tfMess.setText("");
618. **return** text;
619. }
620. @Override
621. **public** **void** handleString(String string) {

624. StyledDocument doc = (StyledDocument) taShow.getDocument();
625. Style style = doc.addStyle("Sty", **null**);
626. **int** start = doc.getLength();
627. **try** {
628. doc.insertString(doc.getLength(), string, style);
629. } **catch** (BadLocationException e) {
630. // **TODO** Auto-generated catch block
631. e.printStackTrace();
632. }
633. **for** (String emoticon : emoticons) {//for each emoticon
634. **int** i = string.indexOf(emoticon);
635. **while** (i >= 0) {
636. **final** SimpleAttributeSet attrs = **new** SimpleAttributeSet(doc.getCharacterElement(start + i).getAttributes());
637. **if** (StyleConstants.*getIcon*(attrs) == **null**) {
638. **switch** (emoticon) {//check which emtoticon picture to apply
639. **case** strAngry:
640. StyleConstants.*setIcon*(attrs, angry);
641. **break**;
642. **case** strBigSmile:
643. StyleConstants.*setIcon*(attrs, big\_smile);
644. **break**;
645. **case** strCry:
646. StyleConstants.*setIcon*(attrs, cry);
647. **break**;
648. **case** strCurlyLips:
649. StyleConstants.*setIcon*(attrs, curly\_lips);
650. **break**;
651. **case** strGlasses:
652. StyleConstants.*setIcon*(attrs, glasses);
653. **break**;
654. **case** strHeart:
655. StyleConstants.*setIcon*(attrs, heart);
656. **break**;
657. **case** strKiki:
658. StyleConstants.*setIcon*(attrs, kiki);
659. **break**;
660. **case** strKiss:
661. StyleConstants.*setIcon*(attrs, kiss);
662. **break**;
663. **case** strLike:
664. StyleConstants.*setIcon*(attrs, like);
665. **break**;
666. **case** strPacman:
667. StyleConstants.*setIcon*(attrs, pacman);
668. **break**;
669. **case** strSad:
670. StyleConstants.*setIcon*(attrs, sad);
671. **break**;
672. **case** strsquinting:
673. StyleConstants.*setIcon*(attrs, squinting);
674. **break**;
675. **case** strTongueOut:
676. StyleConstants.*setIcon*(attrs, tongue\_out);
677. **break**;
678. **case** strUnsure:
679. StyleConstants.*setIcon*(attrs, unsure);
680. **break**;
681. **case** strWink:
682. StyleConstants.*setIcon*(attrs, wink);
683. **break**;
684. **case** strWtf:
685. StyleConstants.*setIcon*(attrs, wtf);
686. **break**;
687. }
689. **try** {
690. doc.remove( start + i, emoticon.length());
691. doc.insertString( start + i, emoticon, attrs);
692. } **catch** (BadLocationException e) {
693. // **TODO** Auto-generated catch block
694. e.printStackTrace();
695. }
696. }
697. i = string.indexOf(emoticon, i + emoticon.length());
698. }
699. }
700. }
701. @Override
702. **public** **void** handleReceiver(String string) {
703. // **TODO** Auto-generated method stub
704. **if** (string.startsWith("<file>")){
705. String [] split = string.split(">");
706. JButton btn = **new** JButton(split[1]);
707. btn.addActionListener(**new** ActionListener(){
708. @Override
709. **public** **void** actionPerformed(ActionEvent arg0) {
710. // **TODO** Auto-generated method stub
711. MainFormController.*receiveFile*(btn.getText());
712. }
714. });
715. **this**.taShow.setCaretPosition(taShow.getDocument().getLength());
716. **this**.taShow.insertComponent(btn);
717. }
718. **else**
719. handleString(string);
720. }
722. **public** **void** resetList(Conversation[] conv) {
723. // **TODO** Auto-generated method stub
724. DefaultListModel model = (DefaultListModel) myList.getModel();
725. model.clear();
726. **for** (**int** i = 0; i < conv.length; i++){
727. **if** (conv[i] == **null**)
728. **break**;
729. model.addElement(conv[i]);
730. }
731. myList.setModel(model);
732. myList.ensureIndexIsVisible(23);
733. myList.setCellRenderer(**new** ConvRenderer());
734. }
736. **public** **void** addActionListenerForJListInMainForm(ListSelectionListener listener) {
737. // **TODO** Auto-generated method stub
738. myList.addListSelectionListener(listener);
739. }
740. @Override
741. **public** String getSelectJList() **throws** Exception {
742. // **TODO** Auto-generated method stub
743. Conversation obj = (Conversation)myList.getSelectedValue();
744. **return** obj.getName();
745. }
747. **public** **void** clearJTextPane(){
748. **this**.taShow.setText("");
749. }
751. **public** String getJButtonTemp(){
752. **return** **this**.btnTemp.getText();
753. }
755. **public** **void** setScrollBottom(){
756. scrollShow.getViewport().setViewPosition(**new** Point(0, taShow.getDocument().getLength()));
757. }
758. }
759. **Client.controller.MainFormController**
760. **package** Client.controller;
761. **import** java.awt.Color;
762. **import** java.awt.Component;
763. **import** java.awt.event.ActionEvent;
764. **import** java.awt.event.ActionListener;
765. **import** java.io.File;
766. **import** java.io.FileOutputStream;
767. **import** java.io.IOException;
768. **import** java.io.ObjectInputStream;
769. **import** java.io.ObjectOutputStream;
770. **import** java.net.Socket;
771. **import** java.nio.file.Files;
772. **import** java.nio.file.Path;
773. **import** java.nio.file.Paths;
774. **import** java.util.Date;
775. **import** javax.swing.ImageIcon;
776. **import** javax.swing.JButton;
777. **import** javax.swing.JFileChooser;
778. **import** javax.swing.JOptionPane;
779. **import** javax.swing.SwingUtilities;
780. **import** javax.swing.SwingWorker;
781. **import** javax.swing.event.ListSelectionEvent;
782. **import** javax.swing.event.ListSelectionListener;
783. **import** javax.swing.text.StyledDocument;
784. **import** javax.swing.text.Utilities;
785. **import** Client.view.IMainForm;
786. **import** Client.view.MainForm;
787. **import** Server.DataSend;
788. **import** Client.model.\*;
789. **public** **class** MainFormController {
790. **public** IMainForm mainForm;
791. **public** **static** String *userName*;
792. **public** **static** String *friendUserName*;
793. **private** Conversation [] conv = **new** Conversation[100];
794. **private** **boolean** stopThread = **false**;
795. **public** MainFormController(String userName) **throws** InterruptedException{
796. mainForm = **new** MainForm(userName);
797. **this**.*userName* = userName;
798. *friendUserName* = " "+userName+" ";
799. mainForm.addActionListenerForNewMessageInMainForm(**new** ActionListener(){
800. @Override
801. **public** **void** actionPerformed(ActionEvent arg0) {
802. // **TODO** Auto-generated method stub
803. **new** NewMessageFormController();
804. }
805. });
807. mainForm.addActionListenerForChangeAvatarInMainForm(**new** ActionListener(){
808. @Override
809. **public** **void** actionPerformed(ActionEvent arg0) {
810. // **TODO** Auto-generated method stub
811. JFileChooser obj = **new** JFileChooser();
812. **int** returnVal = obj.showOpenDialog((Component) mainForm);
813. **if** (returnVal == JFileChooser.***APPROVE\_OPTION***) {
814. File file = obj.getSelectedFile();
815. **if** (!(file.getName().contains(".jpg") || file.getName().contains(".png") || file.getName().contains(".gif")))
816. JOptionPane.*showMessageDialog*(**null**, "Invalid image format!", "Error", 0);
817. **else**{
818. Path path = Paths.*get*(file.getPath());
819. **try** {
820. **byte** [] data = Files.*readAllBytes*(path);
821. DataSend dataSend = **new** DataSend("changeAvatar", userName, **null**, "|"+*friendUserName*+"|", **null**, userName, data);
822. Socket clientSocket = **new** Socket("localhost", 5000);
823. ObjectOutputStream out = **new** ObjectOutputStream(clientSocket.getOutputStream());
824. out.writeObject(dataSend);
825. out.flush();
826. **if** (clientSocket.isInputShutdown() || clientSocket.isOutputShutdown())
827. clientSocket.close();
828. }
829. **catch** (IOException e) {
830. // **TODO** Auto-generated catch block
831. e.printStackTrace();
832. }
833. }
834. }
835. }
837. });
839. mainForm.addActionListenerForJListInMainForm(**new** ListSelectionListener(){
840. @Override
841. **public** **void** valueChanged(ListSelectionEvent arg0) {
842. // **TODO** Auto-generated method stub
843. **try** {
844. **if** (mainForm.getSelectJList()!= **null**){
845. **try** {
846. *friendUserName* = " "+mainForm.getSelectJList()+" ";
847. } **catch** (Exception e) {
848. // **TODO** Auto-generated catch block
849. e.printStackTrace();
850. }
851. System.***out***.println(*friendUserName*);
852. }
853. } **catch** (Exception e) {
854. // **TODO** Auto-generated catch block
855. e.printStackTrace();
856. }
857. }
859. });
861. mainForm.addActionListenerForSignOutInMainForm(**new** ActionListener(){
862. @Override
863. **public** **void** actionPerformed(ActionEvent arg0) {
864. // **TODO** Auto-generated method stub
865. //close 2 thread
866. mainForm.closeForm();
867. stopThread = **true**;
868. SignInController signIn = **new** SignInController();
870. }
871. });
872. mainForm.addActionListenerForSendButtonInMainForm(**new** ActionListener(){
873. @Override
874. **public** **void** actionPerformed(ActionEvent arg0) {
875. // **TODO** Auto-generated method stub
876. String message = mainForm.getTextToSend().replaceAll("\n", "").replaceAll("\_", "");
877. // mainForm.handleReceiver(text);//, Color.red);
878. DataSend dataSend = **new** DataSend("sendMsg", userName, " ", "|"+*friendUserName*+"|", message, " ", **new** **byte**[2]);
879. **try**{
880. Socket clientSocket = **new** Socket("localhost", 5000);
881. ObjectOutputStream out = **new** ObjectOutputStream(clientSocket.getOutputStream());
882. out.writeObject(dataSend);
883. System.***out***.println("DataSend: "+dataSend.getInfo());
884. **if** (clientSocket.isInputShutdown() || clientSocket.isOutputShutdown())
885. clientSocket.close();
886. }
887. **catch** (IOException e) {
888. // **TODO** Auto-generated catch block
889. e.printStackTrace();
890. }
891. }
892. });
893. mainForm.addActionListenerForAttachFileInMainForm(**new** ActionListener(){
895. @Override
896. **public** **void** actionPerformed(ActionEvent arg0) {
897. // **TODO** Auto-generated method stub
898. JFileChooser obj = **new** JFileChooser();
899. **int** returnVal = obj.showOpenDialog((Component) mainForm);
900. **if** (returnVal == JFileChooser.***APPROVE\_OPTION***) {
901. File file = obj.getSelectedFile();
902. Path path = Paths.*get*(file.getPath());
903. **try** {
904. **byte** [] data = Files.*readAllBytes*(path);
905. DataSend dataSend = **new** DataSend("sendFile", userName, **null**, "|"+*friendUserName*+"|", **null**, file.getName().replaceAll("\_", ""), data);
906. Socket clientSocket = **new** Socket("localhost", 5000);
907. ObjectOutputStream out = **new** ObjectOutputStream(clientSocket.getOutputStream());
908. out.writeObject(dataSend);
909. out.flush();
910. **if** (clientSocket.isInputShutdown() || clientSocket.isOutputShutdown())
911. clientSocket.close();
912. }
913. **catch** (IOException e) {
914. // **TODO** Auto-generated catch block
915. e.printStackTrace();
916. }
917. }
918. }
919. });
920. /////////////
921. Thread updateListChatThread = **new** Thread(){
922. **public** **void** run(){
923. **while** (!stopThread){
924. updateListChat();
925. **try** {
926. *sleep*(2000);
927. } **catch** (InterruptedException e) {
928. // **TODO** Auto-generated catch block
929. e.printStackTrace();
930. }
931. }
932. }
933. };
934. updateListChatThread.start();
935. /////////////
936. Thread updateChatWindowThread = **new** Thread(){
937. **public** **void** run(){
938. **while** (!stopThread){
939. updateChatWindow();
940. **try** {
941. *sleep*(1000);
942. } **catch** (InterruptedException e) {
943. // **TODO** Auto-generated catch block
944. e.printStackTrace();
945. }
946. }
947. }
948. };
949. updateChatWindowThread.start();
950. /////////////

953. }
955. **private** **void** updateListChat(){
956. DataSend dataSend = **new** DataSend("updateListChat", *userName*, " ", " ", " ", " ", **new** **byte**[2]);
957. **try**{
958. Socket clientSocket = **new** Socket("localhost", 5000);
959. ObjectOutputStream out = **new** ObjectOutputStream(clientSocket.getOutputStream());
960. ObjectInputStream in = **new** ObjectInputStream(clientSocket.getInputStream());
961. out.writeObject(dataSend);
962. System.***out***.println("DataSend: "+dataSend.getInfo());
963. DataSend dataReceive = (DataSend) in.readObject();
964. System.***out***.println("DataReceive: "+dataReceive.getInfo());
965. clientSocket.close();
966. String [] yourFriendUserList = dataReceive.getYourFriendUserName().split("\_");
967. String [] messageList = dataReceive.getMessage().split("\_");
968. String [] timeList = dataReceive.getTimelist().split("\_");
969. String [] displayUserList = **new** String [100];
970. **for** (**int** i = 0; i<yourFriendUserList.length; i++){
971. displayUserList[i] = (" "+yourFriendUserList[i]+" ").replaceAll(" "+*userName*+" ", " ").trim();
972. File fileAvatar = **new** File("src/avatar/"+displayUserList[i]);
973. Path path = Paths.*get*("src/avatar/"+displayUserList[i]);
974. **if** (fileAvatar.exists()){
975. Files.*delete*(path);
976. }
977. Socket clientSocket1 = **new** Socket("localhost", 5000);
978. ObjectOutputStream out1 = **new** ObjectOutputStream(clientSocket1.getOutputStream());
979. ObjectInputStream in1 = **new** ObjectInputStream(clientSocket1.getInputStream());
980. DataSend dataSend1 = **new** DataSend("updateAvatar", displayUserList[i], " ", " ", " ", " ", **new** **byte**[2]);
981. out1.writeObject(dataSend1);
982. System.***out***.println("DataSend: "+dataSend1.getInfo());
983. DataSend dataReceive1 = (DataSend) in1.readObject();
984. System.***out***.println("DataReceive: "+dataReceive1.getInfo());
985. clientSocket1.close();
986. **if** (dataReceive1.getCommand().equals("updateAvatarOK")){
987. FileOutputStream fos = **new** FileOutputStream("src/avatar/"+displayUserList[i]);
988. fos.write(dataReceive1.getData());
989. fos.close();
990. }
991. }
993. //Cap nhat avatar tai day
994. **for** (**int** i = 0; i< yourFriendUserList.length; i++){
995. conv[i] = **new** Conversation (yourFriendUserList[i], (" "+yourFriendUserList[i]+" ").replaceAll(" "+*userName*+" ", " ").trim(), messageList[i].replaceAll("\\|", " "), Long.*parseLong*(timeList[i]));
997. }
998. }
999. **catch** (Exception e) {
1000. // **TODO** Auto-generated catch block
1001. e.printStackTrace();
1002. }
1003. mainForm.resetList(conv);
1004. }
1006. **private** **void** updateChatWindow(){
1007. DataSend dataSend;
1008. dataSend = **new** DataSend("updateChatWindow", *userName*, " ", "|"+*friendUserName*+"|", " ", " ", **new** **byte**[2]);
1009. **try**{
1010. Socket clientSocket = **new** Socket("localhost", 5000);
1011. ObjectOutputStream out = **new** ObjectOutputStream(clientSocket.getOutputStream());
1012. ObjectInputStream in = **new** ObjectInputStream(clientSocket.getInputStream());
1013. out.writeObject(dataSend);
1014. System.***out***.println("DataSend: "+dataSend.getInfo());
1015. DataSend dataReceive = (DataSend) in.readObject();
1016. System.***out***.println("DataReceive: "+dataReceive.getInfo());
1017. clientSocket.close();
1018. String [] userSendList = dataReceive.getTimelist().split("\_");
1019. String [] messageList = dataReceive.getMessage().split("\_");
1020. mainForm.clearJTextPane();
1021. **for** (**int** i = 1; i < userSendList.length; i++){
1022. **if** (userSendList[i].equals(*userName*))
1023. mainForm.insertStringInPane(12, userSendList[i]+": ", Color.***red***);
1024. **else** mainForm.insertStringInPane(12, userSendList[i]+": ", Color.***blue***);
1025. mainForm.handleReceiver(messageList[i]);
1026. mainForm.handleString("\n");
1027. }
1028. }
1029. **catch** (Exception e) {
1030. // **TODO** Auto-generated catch block
1031. e.printStackTrace();
1032. }
1033. }
1035. **public** **static** **void** receiveFile (String fileName){
1036. DataSend dataSend = **new** DataSend("receiveFile", *userName*, " ", " ", " ", fileName, **new** **byte**[2]);
1037. **try**{
1038. Socket clientSocket = **new** Socket("localhost", 5000);
1039. ObjectOutputStream out = **new** ObjectOutputStream(clientSocket.getOutputStream());
1040. ObjectInputStream in = **new** ObjectInputStream(clientSocket.getInputStream());
1041. out.writeObject(dataSend);
1042. System.***out***.println("DataSend: "+dataSend.getInfo());
1043. DataSend dataReceive = (DataSend) in.readObject();
1044. System.***out***.println("DataReceive: "+dataReceive.getInfo());
1045. clientSocket.close();
1046. FileOutputStream fos = **new** FileOutputStream("D:\\Samsung\\Chat\\"+fileName);
1047. fos.write(dataReceive.getData());
1048. fos.close();
1049. JOptionPane.*showMessageDialog*(**null**, "File saved to D:\\Samsung\\Chat\\"+fileName);
1050. }
1051. **catch** (IOException | NullPointerException e1) {
1052. // **TODO** Auto-generated catch block
1053. e1.printStackTrace();
1054. } **catch** (ClassNotFoundException e1) {
1055. // **TODO** Auto-generated catch block
1056. e1.printStackTrace();
1057. }
1058. }
1059. }

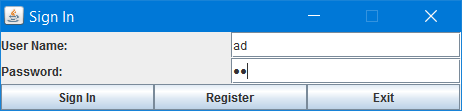
**PHẦN V: KIỂM TRA CHƯƠNG TRÌNH**

1. **Giao diện chính**



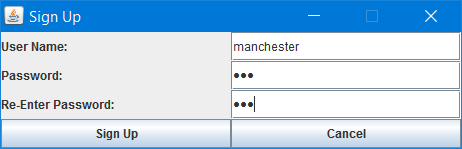
*Hình 10: Giao diện chính*

1. **Giao diện đăng nhập**



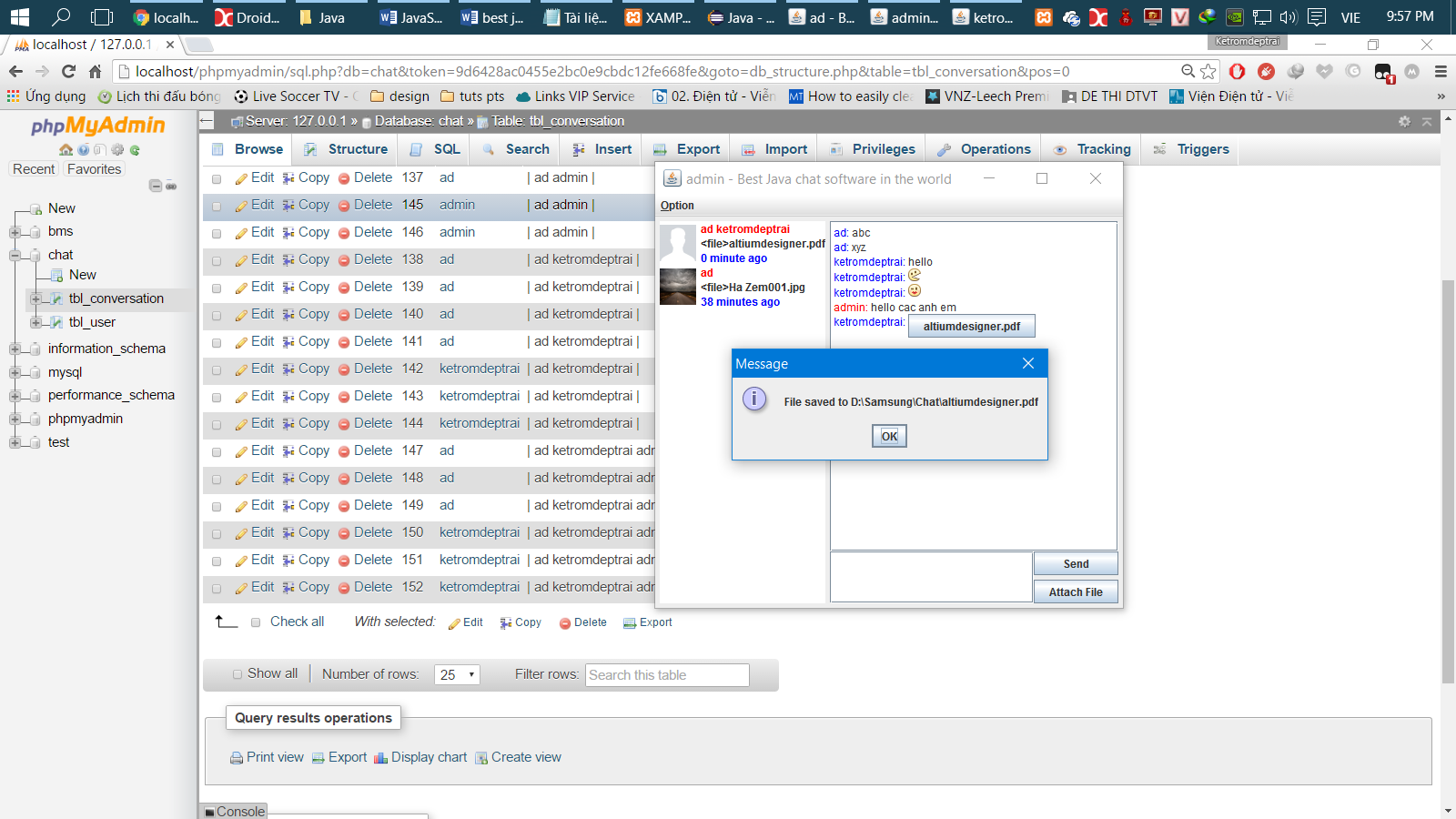
*Hình 11: Giao diện đăng nhập*

1. **Giao diện đăng kí**



*Hình 12: Giao diện đăng kí*

***Sau khi kiểm tra chương trình, nhóm nhận thấy phần mềm hoạt động khá ổn định, các tính năng được kiểm tra đều đạt yêu cầu***



*Hình 13: Thử nghiệm tính năng gửi nhận file*

**PHẦN VI: KẾT LUẬN**

*Qua việc hoàn thành bài tập lớn lần này, chúng em đã học hỏi được thêm nhiều kinh nghiệm trong việc thiết kế và hoàn thiện một phần mềm hoàn chỉnh bằng ngôn ngữ lập trình Java. Cùng với đó, chúng em còn được nâng cao, bổ sung thêm nhiều kiến thức hay và mới về ngôn ngữ này cũng như có cải thiện được kĩ năng mềm của bản thân*

*Em xin chân thành cảm ơn thầy đã giúp đỡ chúng em hoàn thành tốt bài tập lớn nói riêng và môn học này nói chung!*

**TÀI LIỆU THAM KHẢO**