

## 1. How to run

First, run com/server/server.java in server module.

Second, run com/Main.java in client module.

## 2. Explanation

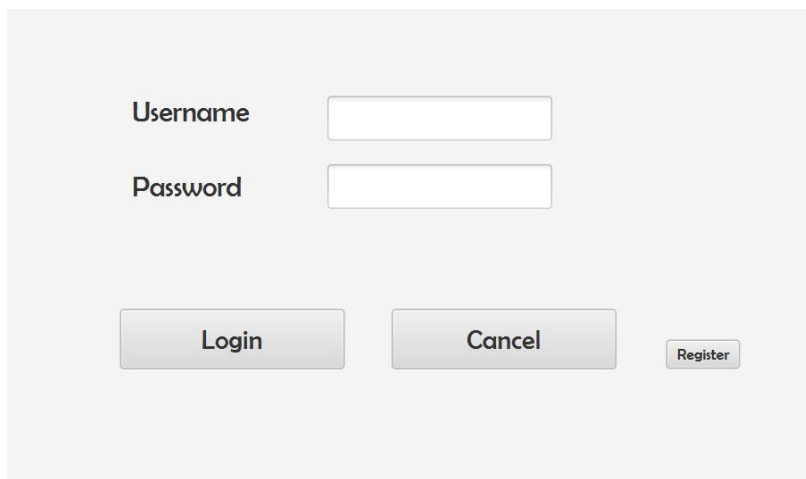
In server side, I use multi threading to implement the function of connecting to different clients.

```
class ClientHandler extends Thread {...}
```

In client side, I use Main.java extends Application and launch it to open the primary stage.

```
try {
    Parent root = FXMLLoader.load(Main.class.getResource( name: "login/loginView.fxml"));
    primaryStage.initStyle(StageStyle.UNDECORATED);
    primaryStage.setTitle("Login Window");
    Scene mainScene = new Scene(root);
    mainScene.setRoot(root);
    primaryStage.setResizable(true);
    primaryStage.setScene(mainScene);
    primaryStage.show();
}catch (IOException e) {
    e.printStackTrace();
}
```

And this is the login view:



Press login button, the system will connect the database in MySQL, and check the username and password. If there is nothing wrong, it will pop up the chat view and close the login view at the same time. Otherwise, an error message is displayed.

```
try {
    JdbcUtils jdbcUtils = new JdbcUtils();
    Connection con= jdbcUtils.getConnection();
    String result = jdbcUtils.search(name,pass,con);
    if (result!=""){ // Has found the information of this user!
        LoginMessage.setVisible(false);
        FXMLLoader loader= new FXMLLoader();
        loader.setLocation(Main.class.getResource( name: "chat/ChatWindow.fxml"));
        Parent root = loader.load();
        showScene( title: "Chat Window",root);
        ChatController controller = loader.getController();
        Listener listener = new Listener(name,port,ip,controller);
        Thread x = new Thread(listener);
        x.start();
        stage= (Stage) cancelButton.getScene().getWindow();
        stage.close(); //The login window should be closed at the same time.
    }else{
        LoginMessage.setVisible(true);
    }
    con.close();
}
```

And in here: JdbcUtils is a class defined by myself. (com.util.JdbcUtils)

```
1 usage new
public JdbcUtils() {
    try {
        Class.forName( className: "com.mysql.cj.jdbc.Driver");
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

The method getConnection() will return a Connection with MySQL:

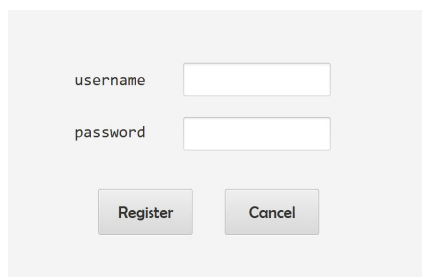
```
public Connection getConnection() {
    try {
        connection = DriverManager.getConnection( url: "jdbc:mysql://localhost:3306/chatroom?serverTimezone=UTC", user: "root", password: "900420");
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return connection;
}
```

The method search() is used to check whether the username and password are correct.

```
1 usage new *
public String search(String name, String pass, Connection con) throws SQLException { //A method to search accounts.
    String result= "";
    Statement sql = con.createStatement();
    ResultSet rs = sql.executeQuery( sql: "select * from userinfo;");
    while(rs.next()){
        if (rs.getString( columnIndex: 1).equals(name)){ //Check the username first. Because username is unique.
            if (rs.getString( columnIndex: 2).equals(pass)){ //Then verify the accuracy of password.
                result= rs.getString( columnIndex: 4); //Attain the head portrait.
                break;
            }
        }
    }
    rs.close();
    sql.close();
    con.close();
    return result;
}
```

Press register button, it will pop up a new window for user to register.

The new window :



username

password

You can type down the username and password, when you press "Register", it will create a new connection with the database, and firstly check whether there is a duplicate username. if not, it will execute a sql command, and add a new record.

After registering successfully, it will remind you to press cancel to turn back, so that you can log in with your new account.

This is the method which returns a boolean value to check duplicate name.

```
public boolean duplicationName(String name, Connection con) throws SQLException{
    boolean r = true;
    Statement sql = con.createStatement();
    ResultSet rs = sql.executeQuery( sql: "select * from userinfo;");
    while(rs.next()){
        if (rs.getString( columnIndex: 1).equals(name)){
            r = false;
            break;
        }
    }
    rs.close();
    sql.close();
    return r;
}
```

While the chat window shows up, client will create a new Listener which implements Runnable, by which all clients can chat together.

In Listener, I override the run() method. First I make a socket connection with the sever:

```
try {
    socket = new Socket(hostname,port);
    outputStream = socket.getOutputStream();
    output = new ObjectOutputStream(outputStream);
    inputStream = socket.getInputStream();
    input = new ObjectInputStream(inputStream);
    logger.info("client "+username+" got connected.");
} catch (UnknownHostException e) {
    throw new RuntimeException(e);
} catch (IOException e) {
    throw new RuntimeException(e);
}

public server(int port) throws IOException {
    ServerSocket ss = new ServerSocket(port);
    try {
        while(true){
            Socket s = ss.accept();
            logger.info("The sever is running.");
            Thread t = new ClientHandler(s);
            t.start();
        }
    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        ss.close();
        logger.info("Server socket is closed. ");
    }
}
```

```

try {
    Message firstMessage = new Message(); //create
    firstMessage.setType(MessageType.INFO);
    firstMessage.setName(username);
    output.writeObject(firstMessage);

    Message initMsg= initChatWindow();
    new *
    Platform.runLater(new Runnable() {
        new *
        @Override
        public void run() {
            try {
                controller.setUsernameLabel(initMsg)
            } catch (IOException e) {
                throw new RuntimeException(e);
            }
            controller.setUserList(initMsg);
        }
    });
}

```

firstMessage : created by client, send it to the server to tell server who the client is.

Message: I use Message to send objectInputStream or objectOutputStream.

initMsg: use it to initialize the chat window, such as show the username on the left-top and load all-users-list on the left side.

If you press the send button:

```

public void sendButton() throws IOException{
    String msg= msgBox.getText();
    if(!msgBox.getText().isEmpty()){
        Listener.send(msg);
        msgBox.clear();
    }
}

```

It will call send() method in Listener, which is used to create a new Message instance and send it to the objectoutputstream.

If you press the voice button:

```

public void voiceButton() throws IOException {
    String audioName = "test"+number+".wav";
    AudioRecorder audioRecorder = new AudioRecorder(audioName);
    if (t){
        audioRecorder.start();
        voiceRecord.setText("recording");
    }else {
        audioRecorder.stopRecording();
        number++;
        Listener.sendVoiceMessage(audioName);
        voiceRecord.setText("Voice");
    }
    t=!t;
}

```

It will start recording audio and save it in the project's directory with the specified file name. It should be reminded that the button needs to be pressed twice to complete a recording.

```

3 usages new *
private void write(Message msg) throws IOException {
    for (ObjectOutputStream writer : writers) {
        writer.writeObject(msg);
        writer.reset();
    }
}

```

In server, use write() method to receive the message sent from client, and then send it back to all clients. So in Listener, after reading different messages, the system performs different operations according to different types of

messages.

Now let me make an explanation about the Message Type.

```
case USER:
    controller.addToChat(message);
    break;
```

USER: it means that the message is from client while chatting, message.getName() can return the user's name, and message.getMsg() can return the content of chat.

```
case VOICE:
    controller.addToChat(message);
    break;
```

VOICE: it means that this is a voice message. Method getName() can return the name of the corresponding audio file.

```
case SERVER:
    controller.addAsServer(message);
    break;
```

SERVER: it means that this is the message from server. Such type of message will be created while joining the chat or after leaving the chat.

The method addChat() is in chatController. Its function is to display users' chat messages in the form of bubbles on the screen.

```
if (msg.getName().equals(usernameLabel.getText())) {
    Thread t2 = new Thread(yourMessages);
    t2.setDaemon(true);
    t2.start();
} else {
    Thread t = new Thread(othersMessages);
    t.setDaemon(true);
    t.start();
}
```

It first determines if the user sending the message is the current user, and if so, it creates a new thread(yourMessages). Otherwise it will create a new thread(otherMessages).

```
Task<HBox> othersMessages = () -> {
    BubbledLabel bl = new BubbledLabel();
    if (msg.getType() == MessageType.VOICE){
        bl.setText(msg.getName()+" : Sent a voice message!");
        AudioRecorder audioRecorder= new AudioRecorder(msg.getMsg());
        audioRecorder.play(msg.getMsg());
    }else {
        bl.setText(msg.getName() + " : " + msg.getMsg());
    }
    bl.setBackground(new Background(new BackgroundFill(Color.LIGHTSKYBLUE, radii: null, insets: null)));
    HBox x = new HBox();
    bl.setBubbleSpec(BubbleType.LEFT_CENTER);
    x.getChildren().addAll(bl);

    return x;
};
```

Each thread will judge the message type. Here I import com.messages.Bubble.\*

▼ Bubble

- Bubble
- BubbledLabel
- BubbleType

The origin author is here is the GitHub link to the original project:

<https://github.com/DomHeal/JavaFX-Chat.git>



```

public synchronized void addAsServer(Message message){
    Task<HBox> task = () -> {
        BubbledLabel serverMsg = new BubbledLabel();
        if (message.getMsg().equals("init")){
            serverMsg.setText(message.getName()+" has joined the chat.");
        }else {
            serverMsg.setText(message.getName()+" has left the chat room.");
        }
        serverMsg.setBackground(new Background(new BackgroundFill(Color.MEDIUMPURPLE, null, insets: null)));
        HBox x =new HBox();
        serverMsg.setBubbleSpec(BubbleType.BOTTOM);
        x.setAlignment(Pos.CENTER);
        x.getChildren().addAll(serverMsg);
        return x;
    };
    task.setOnSucceeded(event -> {
        chatPane.getItems().add(task.getValue());
    });
    Thread t = new Thread(task);
    t.setDaemon(true);
    t.start();
}

```

The method addAsServer() is used to display the message sent by server on the chat interface.

```

public void cancelButtonOnAction(ActionEvent event) throws IOException {
    Stage stage = (Stage) close.getScene().getWindow();
    stage.close();
    Message msg = new Message();
    msg.setName(usernameLabel.getText());
    msg.setType(MessageType.SERVER);
    msg.setMsg("close");
    Listener.closeChat();
}

```

```

public void cancelButtonOnAction(ActionEvent event){
    stage= (Stage) cancelButton.getScene().getWindow();
    stage.close();
}

```

```

public void cancel(ActionEvent event) throws SQLException {
    close(con);
    Stage stage = (Stage) cancel.getScene().getWindow();
    stage.close();
}

```

And each cancel button is used to close the window.