File - /Users/s/IdeaProjects/ChatApplication/src/main/java/com/app/chat/ChatClientGUI.java

```
 1 package com.app.chat;
 2
 3 import javax.swing.*;
 4 import java.awt.*;
 5 import java.awt.event.ActionEvent;
 6 import java.awt.event.ActionListener;
 7 import java.io.*;
 8 import java.net.Socket;
 9
10 public class ChatClientGUI {
11     private JFrame frame;
12     private JTextField textField;
13     private JTextArea textArea;
14     private ObjectOutputStream outputStream;
15     private ObjectInputStream inputStream;
16     private String username;
17     private Socket socket;
18
19     public ChatClientGUI() {
20         frame = new JFrame("Chat Client");
21         textField = new JTextField();
22         textArea = new JTextArea();
23
24         frame.setLayout(new BorderLayout());
25         frame.add(textField, BorderLayout.SOUTH);
26         frame.add(new JScrollPane(textArea),
   BorderLayout.CENTER);
27
28         textField.addActionListener(new
   ActionListener() {
29             @Override
30             public void actionPerformed(ActionEvent
    e) {
31                 String text = textField.getText();
32                 if (!text.isEmpty()) {
33                     try {
34                         outputStream.writeObject(
   new ChatMessage(username, text));
35                         textField.setText("");
36                     } catch (IOException ex) {
37                         ex.printStackTrace();
```

```
38                      }
39                  }
40              }
41          });
42
43          frame.setSize(400, 300);
44          frame.setDefaultCloseOperation(JFrame.
    EXIT_ON_CLOSE);
45          frame.setVisible(true);
46      }
47
48      public void connectToServer(String host, int
    port, String username, String password) {
49          try {
50              this.username = username;
51              socket = new Socket(host, port);
52              outputStream = new ObjectOutputStream(
    socket.getOutputStream());
53              inputStream = new ObjectInputStream(
    socket.getInputStream());
54
55              // Send login message
56              outputStream.writeObject(new
    LoginMessage(username, password));
57
58              new Thread(new MessageReceiver()).start
    ();
59          } catch (IOException e) {
60              e.printStackTrace();
61          }
62      }
63
64      private class MessageReceiver implements
    Runnable {
65          @Override
66          public void run() {
67              try {
68                  while (true) {
69                      Message message = (Message)
    inputStream.readObject();
70                      textArea.append(message.
```

```
70 getSender() + ": " + message.getContent() + "\n");
71                 }
72             } catch (IOException |
   ClassNotFoundException e) {
73                 e.printStackTrace();
74             }
75         }
76     }
77
78     public static void main(String[] args) {
79         ChatClientGUI client = new ChatClientGUI
   ();
80         String username = JOptionPane.
   showInputDialog("Enter username:");
81         String password = JOptionPane.
   showInputDialog("Enter password:");
82         client.connectToServer("localhost", 8080,
   username, password);
83     }
84 }
85
```

```java
package com.app.chat;

public class ChatMessage extends Message {
    private final String content;

    public ChatMessage(String sender, String content) {
        super(sender);
        this.content = content;
    }

    @Override
    public String getContent() {
        return content;
    }
}

```

```java
package com.app.chat;

import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.ArrayList;
import java.util.List;

public class ChatServer {
    private final int port;
    private final List<ClientHandler> clients = new ArrayList<>();
    private final DatabaseManager databaseManager = new DatabaseManager();

    public ChatServer(int port) {
        this.port = port;
    }

    public void start() {
        try (ServerSocket serverSocket = new ServerSocket(port)) {
            System.out.println("Server is running on port " + port);
            while (true) {
                Socket clientSocket = serverSocket.accept();
                System.out.println("New client connected: " + clientSocket);
                ClientHandler clientHandler = new ClientHandler(clientSocket, this);
                clients.add(clientHandler);
                clientHandler.start();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public void broadcastMessage(Message message) {
        for (ClientHandler client : clients) {
```

```
35              client.sendMessage(message);
36          }
37      }
38
39      public void sendPrivateMessage(
   PrivateChatMessage message) {
40          ClientHandler recipient =
   getClientByUsername(message.getRecipient());
41          if (recipient ≠ null) {
42              recipient.sendMessage(message);
43          }
44      }
45
46      public void removeClient(ClientHandler client
   ) {
47          clients.remove(client);
48          System.out.println("Client disconnected: "
    + client.getUsername());
49          broadcastMessage(new ChatMessage("Server",
   client.getUsername() + " has left the chat"));
50      }
51
52      public ClientHandler getClientByUsername(String
    username) {
53          for (ClientHandler client : clients) {
54              if (client.getUsername().equals(
   username)) {
55                  return client;
56              }
57          }
58          return null;
59      }
60
61      public DatabaseManager getDatabaseManager() {
62          return databaseManager;
63      }
64
65      public static void main(String[] args) {
66          int port = 8080; // Change this to the
   desired port number
67          ChatServer server = new ChatServer(port);
```

```
68          server.start();
69      }
70  }
71
```

```java
package com.app.chat;

import java.io.*;
import java.net.Socket;

public class ClientHandler extends Thread {
    private final Socket clientSocket;
    private final ChatServer server;
    private ObjectInputStream inputStream;
    private ObjectOutputStream outputStream;
    private String username;

    public ClientHandler(Socket socket, ChatServer server) {
        this.clientSocket = socket;
        this.server = server;
    }

    @Override
    public void run() {
        try {
            inputStream = new ObjectInputStream(clientSocket.getInputStream());
            outputStream = new ObjectOutputStream(clientSocket.getOutputStream());

            handleClient();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
                if (inputStream != null) inputStream.close();
                if (outputStream != null) outputStream.close();
                if (clientSocket != null) clientSocket.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
```

```
36      }
37
38      private void handleClient() {
39          try {
40              while (true) {
41                  Message message = (Message)
    inputStream.readObject();
42                  if (message instanceof LoginMessage
    ) {
43                      LoginMessage loginMessage = (
    LoginMessage) message;
44                      username = loginMessage.
    getUsername();
45                      if (server.getDatabaseManager
    ().authenticateUser(loginMessage.getUsername(),
    loginMessage.getPassword())) {
46                          outputStream.writeObject(
    new ChatMessage("Server", "Login successful"));
47                          server.broadcastMessage(new
     ChatMessage("Server", username + " has joined the
    chat"));
48                      } else {
49                          outputStream.writeObject(
    new ChatMessage("Server", "Login failed"));
50                          break;
51                      }
52                  } else if (message instanceof
    ChatMessage) {
53                      server.broadcastMessage(message
    );
54                  } else if (message instanceof
    PrivateChatMessage) {
55                      PrivateChatMessage
    privateMessage = (PrivateChatMessage) message;
56                      server.sendPrivateMessage(
    privateMessage);
57                  }
58              }
59          } catch (EOFException e) {
60              System.out.println("Client disconnected
    : " + username);
```

```
61          } catch (IOException |
    ClassNotFoundException e) {
62              e.printStackTrace();
63          } finally {
64              server.removeClient(this);
65          }
66      }
67
68      public void sendMessage(Message message) {
69          try {
70              outputStream.writeObject(message);
71          } catch (IOException e) {
72              e.printStackTrace();
73          }
74      }
75
76      public String getUsername() {
77          return username;
78      }
79  }
80
```

```java
package com.app.chat;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

public class DatabaseManager {
    private static final String URL = "jdbc:mysql://localhost:3306/chat_app";
    private static final String USER = "root"; // Update with your MySQL username
    private static final String PASSWORD = "2k4k@Benefit"; // Update with your MySQL password

    static {
        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
            throw new IllegalStateException("Failed to load MySQL JDBC driver");
        }
    }

    public Connection getConnection() throws SQLException {
        return DriverManager.getConnection(URL, USER, PASSWORD);
    }

    public boolean registerUser(String username, String password) {
        String query = "INSERT INTO users (username, password) VALUES (?, ?)";
        try (Connection connection = getConnection();
             PreparedStatement preparedStatement = connection.prepareStatement(query)) {
```

```
31              preparedStatement.setString(1, username
   );
32              preparedStatement.setString(2, password
   );
33              int rowsAffected = preparedStatement.
   executeUpdate();
34              return rowsAffected > 0;
35          } catch (SQLException e) {
36              e.printStackTrace();
37              return false;
38          }
39      }
40
41      public boolean authenticateUser(String username
   , String password) {
42          String query = "SELECT * FROM users WHERE
   username = ? AND password = ?";
43          try (Connection connection = getConnection
   ();
44               PreparedStatement preparedStatement =
   connection.prepareStatement(query)) {
45              preparedStatement.setString(1, username
   );
46              preparedStatement.setString(2, password
   );
47              try (ResultSet resultSet =
   preparedStatement.executeQuery()) {
48                  return resultSet.next();
49              }
50          } catch (SQLException e) {
51              e.printStackTrace();
52              return false;
53          }
54      }
55
56      public boolean isAdmin(String username) {
57          String query = "SELECT is_admin FROM users
   WHERE username = ?";
58          try (Connection connection = getConnection
   ();
59               PreparedStatement preparedStatement =
```

```
59 connection.prepareStatement(query)) {
60             preparedStatement.setString(1, username
   );
61             try (ResultSet resultSet =
   preparedStatement.executeQuery()) {
62                 if (resultSet.next()) {
63                     return resultSet.getBoolean("
   is_admin");
64                 }
65             }
66         } catch (SQLException e) {
67             e.printStackTrace();
68         }
69         return false;
70     }
71 }
72
```

```java
package com.app.chat;

public class LoginMessage extends Message {
    private final String username;
    private final String password;

    public LoginMessage(String username, String password) {
        super(username);
        this.username = username;
        this.password = password;
    }

    @Override
    public String getContent() {
        return null; // Login messages do not have a content field
    }

    public String getUsername() {
        return username;
    }

    public String getPassword() {
        return password;
    }
}

```

```java
package com.app.chat;

import java.io.Serializable;

public abstract class Message implements
Serializable {
    private final String sender;

    public Message(String sender) {
        this.sender = sender;
    }

    public String getSender() {
        return sender;
    }

    public abstract String getContent();
}

```

```java
package com.app.chat;

public class PrivateChatMessage extends Message {
    private final String recipient;
    private final String content;

    public PrivateChatMessage(String sender, String recipient, String content) {
        super(sender);
        this.recipient = recipient;
        this.content = content;
    }

    @Override
    public String getContent() {
        return content;
    }

    public String getRecipient() {
        return recipient;
    }
}

```

Documentation

Introduction

The Chat Application is a simple client-server messaging system designed to facilitate real-time communication between multiple clients over a network. It provides a graphical user interface for users to send and receive messages in a chat room environment.

Features

Client-Server Architecture: Utilizes a client-server model where clients connect to a central server to exchange messages.
Graphical User Interface (GUI): Provides an intuitive GUI for users to interact with the chat system.
Login System: Allows users to authenticate themselves with a username and password.
Public Chat: Supports group chat functionality where users can send messages visible to all connected clients.
Private Messaging: Enables users to send private messages to specific clients.
Error Handling: Implements robust error handling to gracefully manage unexpected events and maintain system stability.
Database Integration: Integrates with a MySQL database to store user credentials and authenticate users during login.

Components

1. Server

ChatServer.java: Implements the server-side functionality, including handling client connections, managing chat rooms, and broadcasting messages.

2. Client

ChatClientGUI.java: Implements the client-side GUI, allowing users to interact with the chat system, send messages, and view received messages.

3. Shared

Message.java: Defines the message objects exchanged between clients and the server, including chat messages, login messages, and private messages.

ClientHandler.java: Manages communication between the server and individual clients, handling message routing and client-server interactions.

DatabaseManager.java: Handles database operations, including user authentication and user registration.

Usage

Server Setup:
Run the ChatServer class to start the server on the specified port (default port: 8080).
Ensure that the MySQL database is set up and running.

Client Login:
Launch the ChatClientGUI class to open the client application.
Enter a username and password to log in to the chat system.
Upon successful login, users can begin sending and receiving messages.

Sending Messages:

Type a message in the text input field at the bottom of the GUI.
Press Enter or click the Send button to send the message.
Sent messages will appear in the chat area, visible to all connected clients.
Private Messaging:

To send a private message, prefix the message with the recipient's username followed by a colon (e.g., recipientUsername: Your private message).
Private messages will only be visible to the sender and the recipient.

Dependencies
Java Development Kit (JDK): Required for compiling and running Java code.
MySQL Connector/J: JDBC driver for connecting to the MySQL database.

Conclusion
The Chat Application provides a simple yet effective platform for real-time communication among multiple users. With its intuitive GUI, robust client-server architecture, and support for both public and private messaging, it serves as a versatile solution for various chat-based communication needs.