Table of Contents

Chatting Application

1.1 Introduction

In response to the project requirements, I developed a Java-based socket programming application that represents the client-server communication model. This application has been specially designed for the College of Information Technology student community, providing them with a platform to facilitate the exchange of messages and files among their peers.

1.2 LoginFrame class

Overview

The LoginFrame class is a simple user authentication and registration system with a graphical user interface (GUI). Users can log in using a username and password or register as new users. The system uses JSON files to store user data.

Class Hierarchy

LoginFrame: The main class responsible for the login user interface.

RegistrationFrame: A separate frame for user registration.

LoginFrame Features

Login: Users can enter their username and password to log in.

Show Password: Users can toggle the visibility of the password.

Register: Users can register for a new account by clicking the "REGISTER" button.

Error Handling: The application provides error messages for invalid input and credentials.

RegistrationFrame Features

Registration: Users can register by providing a username and password.

Password Complexity: Passwords must meet complexity requirements (at least 8 characters with uppercase, lowercase, and a digit).

Username Uniqueness: Usernames must be unique; the system checks for existing usernames.

Functionality

The LoginFrame class manages the main login interface. When the application starts, users can log in or click the "REGISTER" button to create a new account.

When users click "LOGIN," the application validates the input fields and checks if the user's credentials match those stored in a JSON file (users.json).

If login is successful, the application opens a chat client (ITClinet) and closes the login window.
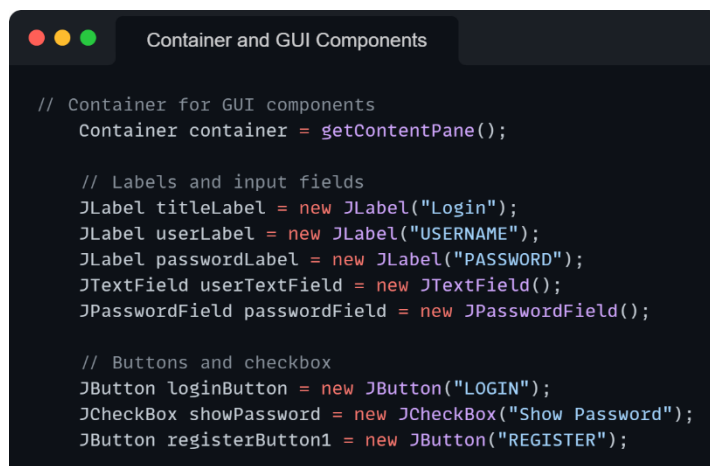
If users click "REGISTER," the RegistrationFrame is displayed, allowing users to register with a unique username and a complex password.

If registration is successful, a new user is added to the JSON file.

- *Details about loginFrame code*
- *LoginFrame class*

Container and GUI Components:

1- The container variable is used to hold the GUI components.
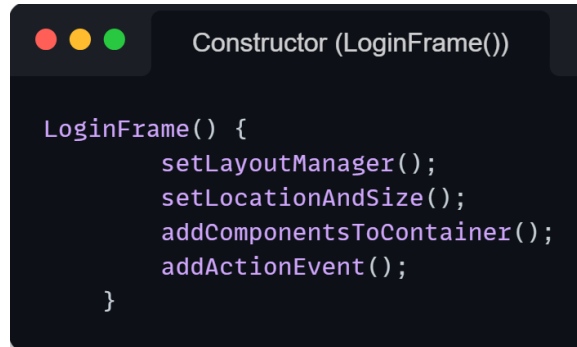2- Labels, text fields, buttons, and a checkbox are defined for the username, password, login, and show password.

```
Container and GUI Components

// Container for GUI components
    Container container = getContentPane();

    // Labels and input fields
    JLabel titleLabel = new JLabel("Login");
    JLabel userLabel = new JLabel("USERNAME");
    JLabel passwordLabel = new JLabel("PASSWORD");
    JTextField userTextField = new JTextField();
    JPasswordField passwordField = new JPasswordField();

    // Buttons and checkbox
    JButton loginButton = new JButton("LOGIN");
    JCheckBox showPassword = new JCheckBox("Show Password");
    JButton registerButton1 = new JButton("REGISTER");
```
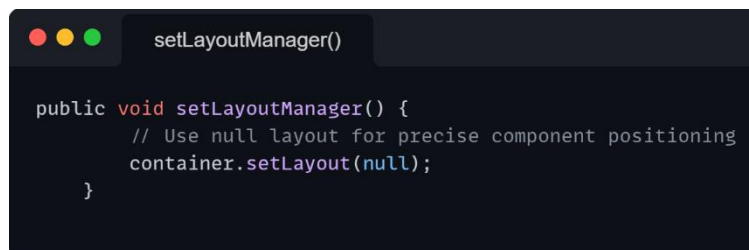
Constructor (LoginFrame()):

1- The constructor sets up the GUI components, layout manager, positions, and sizes.
2- It also adds components to the container and attaches action listeners to the buttons and checkbox

```
Constructor (LoginFrame())

LoginFrame() {
        setLayoutManager();
        setLocationAndSize();
        addComponentsToContainer();
        addActionEvent();
    }
```
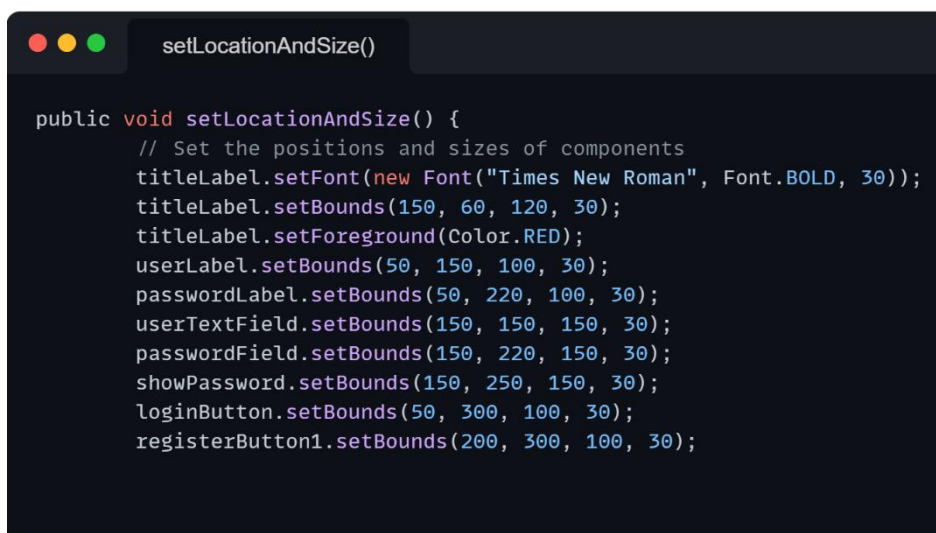
setLayoutManager():

Sets the layout manager to null, allowing for precise component positioning.

```
setLayoutManager()

public void setLayoutManager() {
        // Use null layout for precise component positioning
        container.setLayout(null);
    }
```

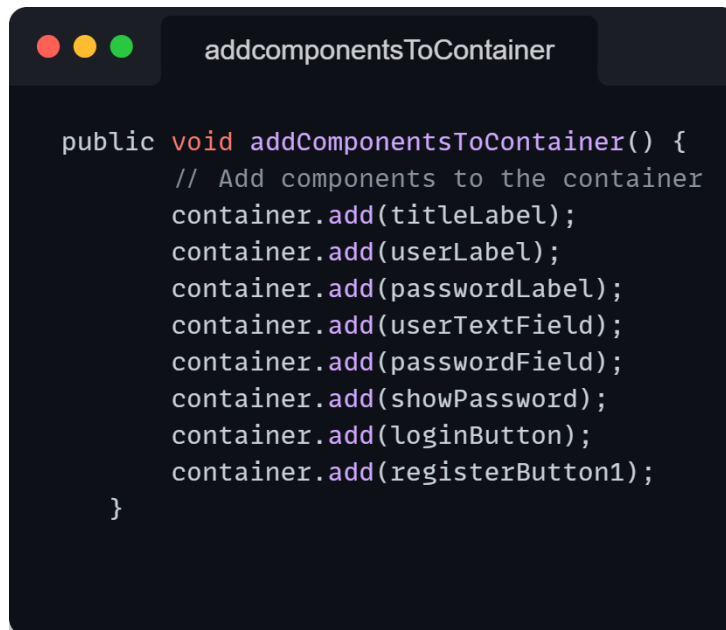### *setLocationAndSize():*

Defines the positions and sizes of GUI components. It also sets the font and color for the title label.

```
setLocationAndSize()

public void setLocationAndSize() {
        // Set the positions and sizes of components
        titleLabel.setFont(new Font("Times New Roman", Font.BOLD, 30));
        titleLabel.setBounds(150, 60, 120, 30);
        titleLabel.setForeground(Color.RED);
        userLabel.setBounds(50, 150, 100, 30);
        passwordLabel.setBounds(50, 220, 100, 30);
        userTextField.setBounds(150, 150, 150, 30);
        passwordField.setBounds(150, 220, 150, 30);
        showPassword.setBounds(150, 250, 150, 30);
        loginButton.setBounds(50, 300, 100, 30);
        registerButton1.setBounds(200, 300, 100, 30);
```
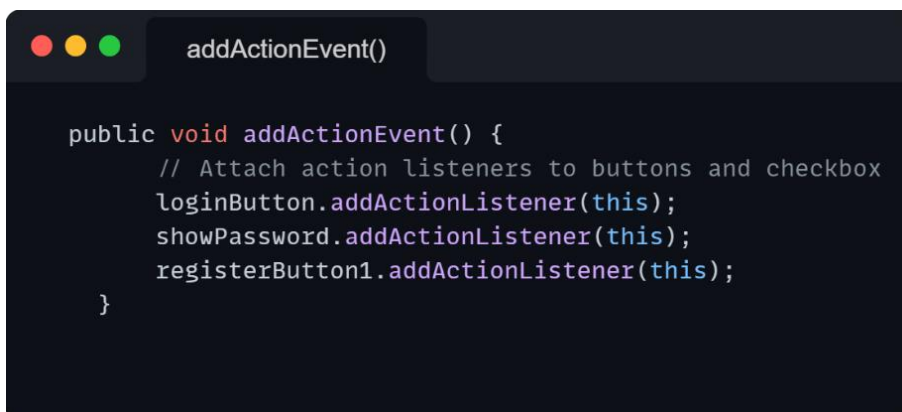
addComponentsToContainer():

Adds GUI components to the frame's container.

```
addcomponentsToContainer

public void addComponentsToContainer() {
        // Add components to the container
        container.add(titleLabel);
        container.add(userLabel);
        container.add(passwordLabel);
        container.add(userTextField);
        container.add(passwordField);
        container.add(showPassword);
        container.add(loginButton);
        container.add(registerButton1);
    }
```

addActionEvent():

Attaches action listeners to the login button, show password checkbox, and register button.

```
addActionEvent()

public void addActionEvent() {
      // Attach action listeners to buttons and checkbox
      loginButton.addActionListener(this);
      showPassword.addActionListener(this);
      registerButton1.addActionListener(this);
   }
```

actionPerformed(ActionEvent e):

1- Handles button clicks and checkbox events.

2- Validates user input, displays error messages or opens a chat client if login is successful.

3- Toggles password visibility when the "Show Password" checkbox is clicked.

4- Opens a registration frame when the "Register" button is clicked.

```java
@Override
public void actionPerformed(ActionEvent e) {
    if (e.getSource() == loginButton) {
        // Handle login button click
        String username = userTextField.getText();
        String password = new String(passwordField.getPassword());
        if (username.isEmpty() || password.isEmpty()) {
            // Show an error message for empty fields
            JOptionPane.showMessageDialog(this, "Please enter a username and password.", "Error", JOptionPane.ERROR_MESSAGE);
        } else if (isUserValid(username, password)) {
            // If login is successful, show a success message and open the chat client
            JOptionPane.showMessageDialog(this, "Login Successful");
            openChatClient();
        } else {
            // Show an error message for invalid credentials
            JOptionPane.showMessageDialog(this, "Invalid Username or Password", "Error", JOptionPane.ERROR_MESSAGE);
        }
    } else if (e.getSource() == showPassword) {
        // Toggle password visibility
        if (showPassword.isSelected()) {
            passwordField.setEchoChar((char) 0);
        } else {
            passwordField.setEchoChar('*');
        }
    } else if (e.getSource() == registerButton1) {
        // Handle registration button click
        RegistrationFrame registrationFrame = new RegistrationFrame(this);
        registrationFrame.setTitle("Registration Form");
        registrationFrame.setVisible(true);
        registrationFrame.setBounds(10, 5, 400, 500);
        registrationFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        registrationFrame.setResizable(true);
    }
}
```
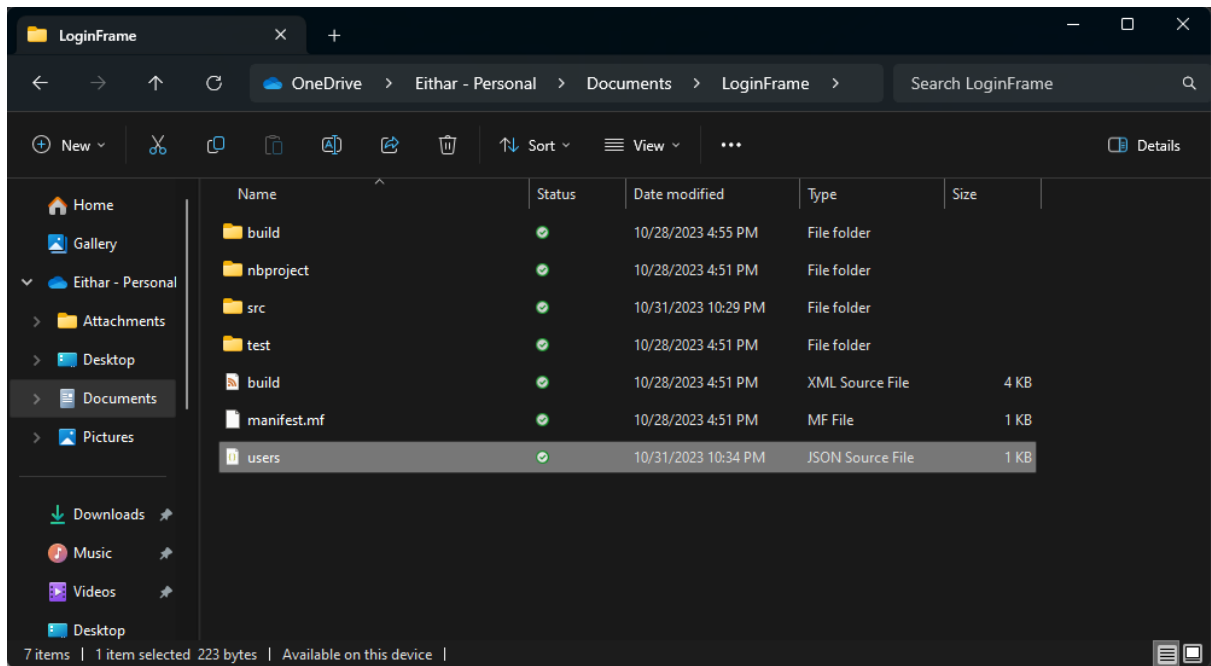
openChatClient():

Opens the chat client when the login is successful.

```java
public void openChatClient() {
        // Open the chat client
        ITClinet chatClient = new ITClinet();
        this.dispose(); // Close the login frame
    }
```

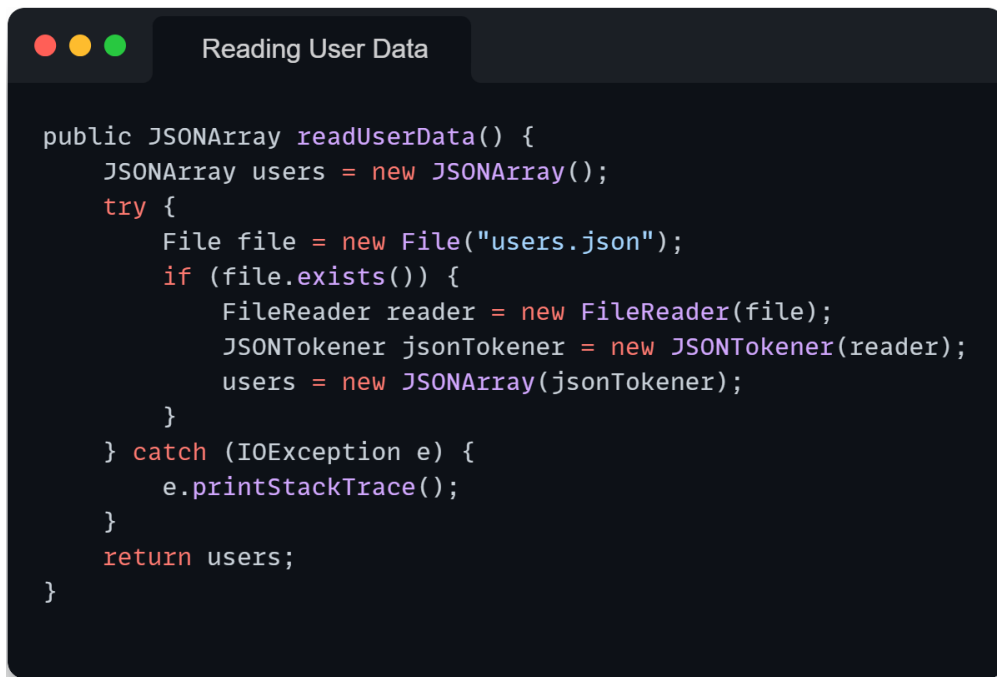- *Details about Saving Users Accounts on the JSON file*

In the login application developed for the College of Information Technology, user account information is stored in a JSON file named "users.json." Below is a detailed explanation of how user accounts are saved to this JSON file

first of all Import Required Libraries

```java
import org.json.JSONArray;
import org.json.JSONObject;
import org.json.JSONTokener;
```

Secondly Reading User Data

-The readUserData method initializes a JSONArray to store user data.

-It checks if the "users.json" file exists.

-If the file exists, it creates a FileReader and a JSONTokener to read the JSON data from the file.

-The JSON data is parsed into the users JSONArray, which contains user account information.

```
public JSONArray readUserData() {
    JSONArray users = new JSONArray();
    try {
        File file = new File("users.json");
        if (file.exists()) {
            FileReader reader = new FileReader(file);
            JSONTokener jsonTokener = new JSONTokener(reader);
            users = new JSONArray(jsonTokener);
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
    return users;
}
```

Thirdly Writing User Data

-The writeUserData method takes a JSONArray (users) as an argument, which contains updated user account information.

-It creates a FileWriter to write data to the "users.json" file.

-The toString(4) method is used to format the JSON data with an indentation of 4 spaces to make it more human-readable.

-The data is written to the file, and the writer is flushed and closed.

```
public void writeUserData(JSONArray users) {
    try {
        FileWriter writer = new FileWriter("users.json");
        writer.write(users.toString(4)); // Use 4 as an argument to pretty-print JSON
        writer.flush();
        writer.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```
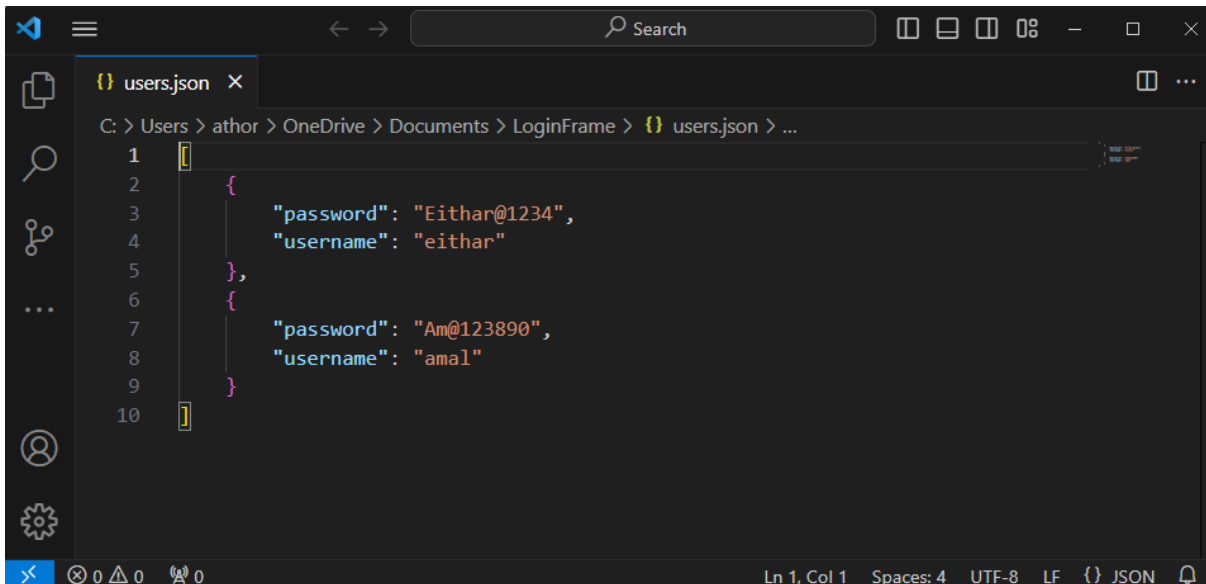
The last thing is Checking User Validity

-The isUserValid method checks if a given username and password are valid by reading user data from the JSON file using readUserData.

-It iterates through the user accounts stored in the users JSONArray.

-If it finds a match between the provided username and password with the stored data,

-it returns true, indicating the user is valid.

-If no match is found, it returns false.

```java
public boolean isUserValid(String username, String password) {
    JSONArray users = readUserData();
    for (int i = 0; i < users.length(); i++) {
        JSONObject user = users.getJSONObject(i);
        if (user.getString("username").equals(username) && user.getString("password").equals(password)) {
            return true;
        }
    }
    return false;
}
```

The data will be save like this

```json
[
    {
        "password": "Eithar@1234",
        "username": "eithar"
    },
    {
        "password": "Am@123890",
        "username": "amal"
    }
]
```

main(String[ ] a):

The main method creates and displays the login frame.

```
main(String[] a)

public static void main(String[] a) {
        // Create and display the login frame
        LoginFrame frame = new LoginFrame();
        frame.setTitle("Login Form");
        frame.setVisible(true);
        frame.setBounds(10, 5, 400, 500);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setResizable(true);
    }
```

RegistrationFrame Class:

These components collectively create a user registration form within a graphical user interface. Users enter their username and password, click the "REGISTER" button to complete the registration process, and the loginFrame reference is used for specific actions related to user registration and updating the data in the login frame.

```
    // Reference to the LoginFrame instance
    private final LoginFrame loginFrame;

    Container container = getContentPane();
    JLabel titleLabel = new JLabel("Register");
    JLabel userLabel = new JLabel("USERNAME");
    JLabel passwordLabel = new JLabel("PASSWORD");
    JTextField userTextField = new JTextField();
    JPasswordField passwordField = new JPasswordField();
    JButton registerButton2 = new JButton("REGISTER");
```

Constructor (RegistrationFrame(LoginFrame loginFrame)):

1- Takes a reference to the LoginFrame instance.
2- Sets up the GUI components for user registration and adds an action listener to the registration button.

```
Constructor (RegistrationFrame(LoginFrame loginFrame))

public RegistrationFrame(LoginFrame loginFrame) {
        this.loginFrame = loginFrame; // Initialize the reference to the LoginFrame
        setLayoutManager();
        setLocationAndSize();
        addComponentsToContainer();
        registerButton2.addActionListener(this);
    }
```

actionPerformed(ActionEvent e):

1- Handles button clicks in the registration frame.
2- Validates user input, displays error messages, or adds a new user to the user data if registration is successful.

```java
@Override
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == registerButton2) {
            // Handle registration button click
            String username = userTextField.getText();
            String password = new String(passwordField.getPassword());

            if (username.isEmpty() || password.isEmpty()) {
                // Show an error message for empty fields
                JOptionPane.showMessageDialog(this, "Please enter a username and password.", "Error", JOptionPane.ERROR_MESSAGE);
            } else if (isUsernameTaken(username)) {
                // Show an error message if the username is already taken
                JOptionPane.showMessageDialog(this, "Username is already taken.", "Error", JOptionPane.ERROR_MESSAGE);
            } else if (!isPasswordValid(password)) {
                // Show an error message for an invalid password
                String errorMessage = "<html> <center> Password does not meet the complexity requirements. <br>" +
                        "It should contain at least 8 characters <br> "
                        + "including uppercase, lowercase, and a digit. <br> <br> "
                        + "Please try again. </center> </html>";
                JOptionPane.showMessageDialog(this,errorMessage, "Error", JOptionPane.ERROR_MESSAGE);
            } else {
                // Read existing user data from the LoginFrame
                JSONArray users = loginFrame.readUserData();

                // Create a new user and save it to the JSON file
                JSONObject newUser = new JSONObject();
                newUser.put("username", username);
                newUser.put("password", password);
                // Add the new user to the array
                users.put(newUser);
                // Write the updated user data back to the JSON file
                loginFrame.writeUserData(users);

                // Show a success message and close the registration frame
                JOptionPane.showMessageDialog(this, "Registration Successful");
                this.dispose();
            }
        }
    }
```

isPasswordValid(String password):

Checks if the password meets complexity requirement

```java
private boolean isPasswordValid(String password) {
        // Check if the password meets complexity requirements
        String regex = "^(?=.*[a-z])(?=.*[A-Z])(?=.*\\d).{8,}$";
        return password.matches(regex);
    }
```

isUsernameTaken(String username):

Checks if the entered username is already in use.

```java
private boolean isUsernameTaken(String username) {
    // Check if the username is already in use
    JSONArray users = loginFrame.readUserData();
    for (int i = 0; i < users.length(); i++) {
        JSONObject user = users.getJSONObject(i);
        if (user.getString("username").equals(username)) {
            return true;
        }
    }
    return false;
}
```

setLayoutManager(), setLocationAndSize(), and addComponentsToContainer():

Set up the layout and position of GUI components in the registration frame.

```java
public void setLayoutManager() {
    // Use null layout for precise component positioning
    container.setLayout(null);
}

public void setLocationAndSize() {
    // Set the positions and sizes of components
    titleLabel.setFont(new Font("Times New Roman", Font.BOLD, 30));
    titleLabel.setBounds(150, 60, 120, 30);
    titleLabel.setForeground(Color.RED);
    userLabel.setBounds(50, 150, 100, 30);
    passwordLabel.setBounds(50, 220, 100, 30);
    userTextField.setBounds(150, 150, 150, 30);
    passwordField.setBounds(150, 220, 150, 30);
    registerButton2.setBounds(125, 300, 100, 30);
}

public void addComponentsToContainer() {
    // Add components to the container
    container.add(titleLabel);
    container.add(userLabel);
    container.add(passwordLabel);
    container.add(userTextField);
    container.add(passwordField);
    container.add(registerButton2);
}
```

1.3 ITClinet Class

Overview
The ITClinet application is a client-side chat client with a graphical user interface (GUI). Users can send messages and select and send files to a server. It receives messages from the server and displays them in the chat window.

Class Hierarchy
ITClinet: The main class responsible for the client-side chat interface.

Features
Send Messages: Users can enter text messages and send them to the another user.

Send Files: Users can select and send files to the another user.

Receive Messages: The application receives messages from the clinte and displays them.

Exception Handling: The client handles exceptions gracefully and closes the connection.

Functionality
When the application starts, it establishes a connection to a chat server (typically running on localhost).

Users can send text messages by entering text and clicking the "Send" button.

Users can send files by clicking the "Browse" button, selecting a file, and clicking "Send."

The application continuously listens for messages from the server and displays them in the chat window.

The chat client runs in a separate thread, ensuring that the GUI remains responsive.

1.4 ITServer Class

Overview

The ITServer application is a chat server that listens for incoming client connections. It establishes connections with clients and manages client handlers to handle message broadcasting.

Class Hierarchy

ITServer: The main class responsible for the chat server functionality.

Features

Accept Connections: The server listens for incoming client connections on a specified port (1248).

Client Handlers: Each connected client is managed by a separate ClientHandler thread.

Broadcast Messages: The server receives messages from clients and broadcasts them to all connected clients.

Graceful Shutdown: The server handles exceptions and gracefully closes connections.

Functionality

The server is started, and it listens for incoming client connections on a specified port (e.g., 1248).

When a client connects, a separate ClientHandler thread is created to manage the client.
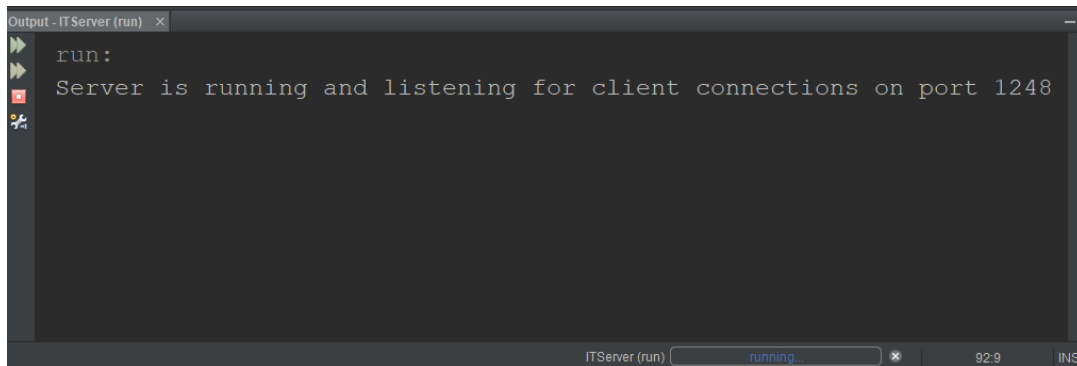
The ClientHandler receives messages from the client and broadcasts them to all connected clients.

The server manages a list of connected clients.

When a client disconnects, its corresponding ClientHandler is removed, ensuring that other clients are not affected.
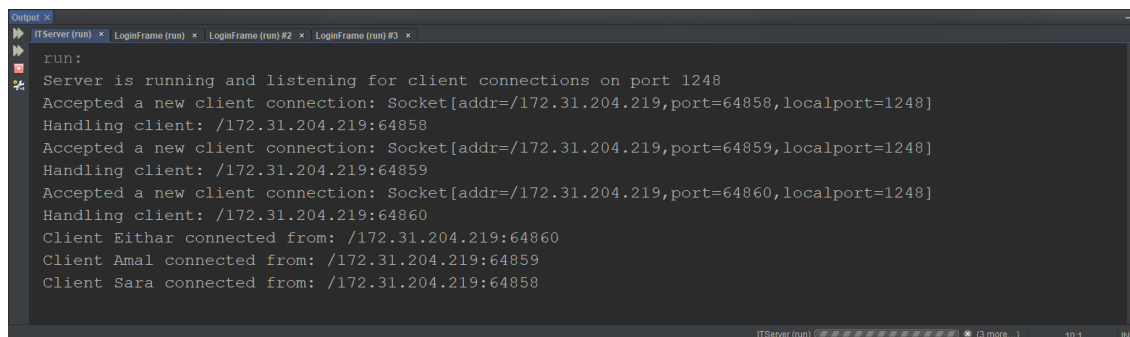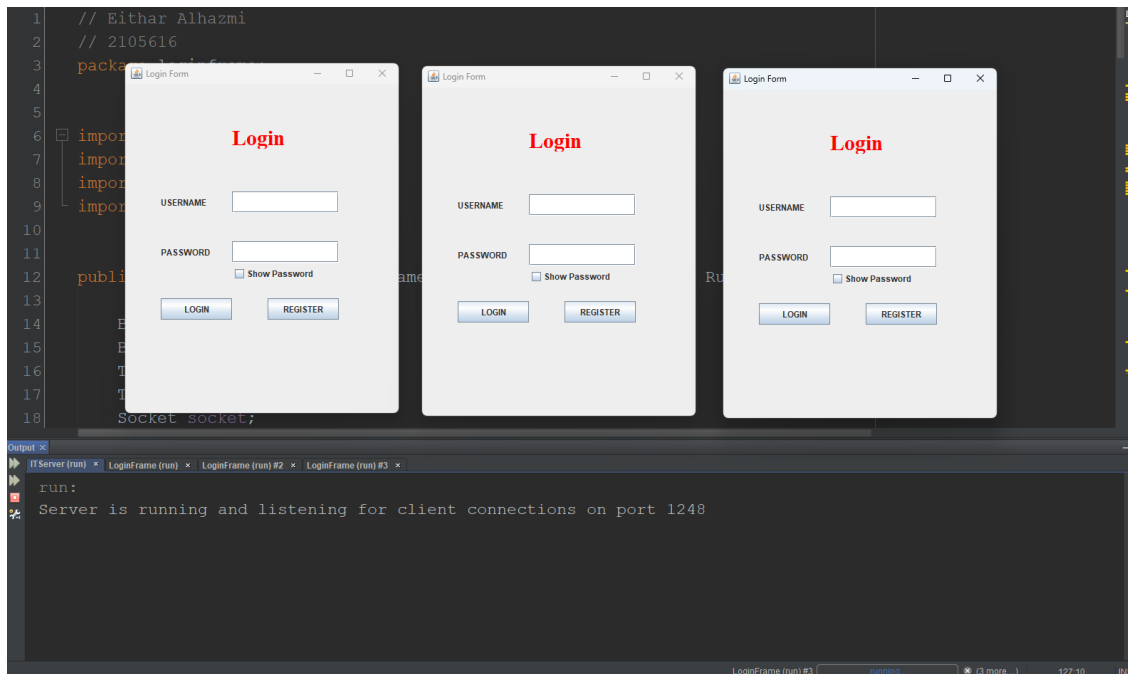
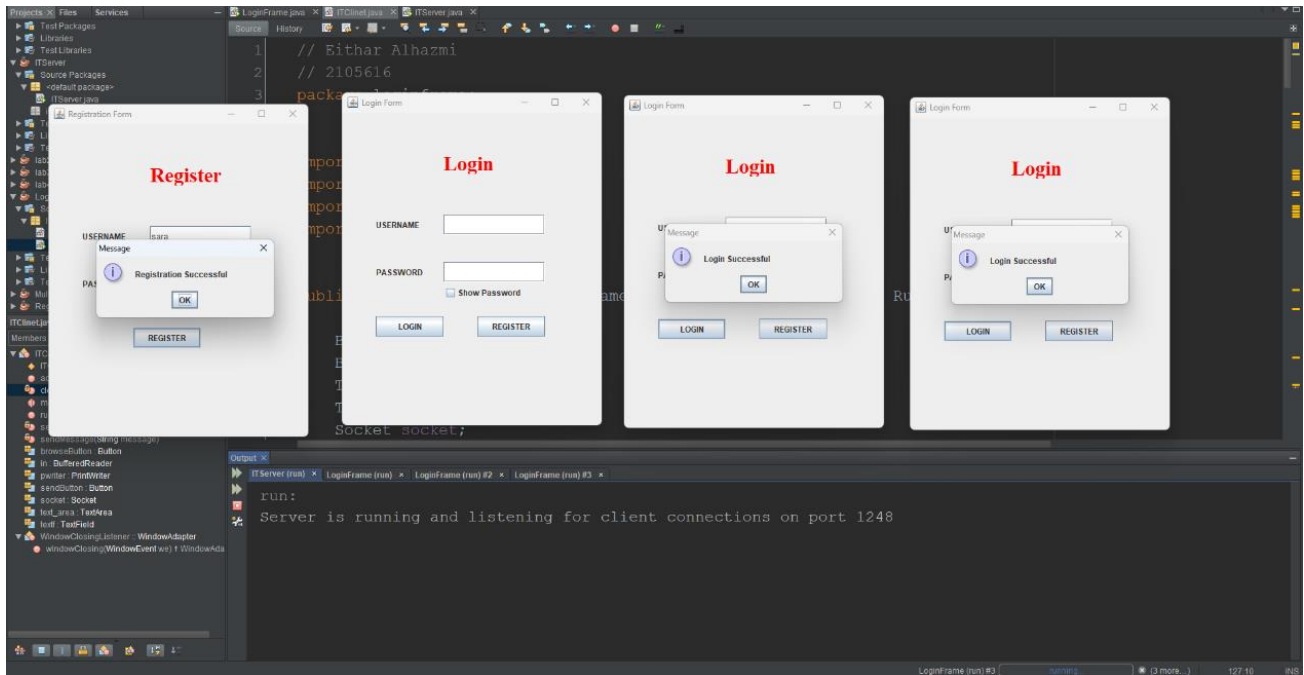- *The Run of Application*

Run The Server on Port 1248

Maintain a list of active client connections and the server be able to handle multiple client connections simultaneously

register with the application using a unique username and password.

Upon launching the application, users are required to input their names. This procedural step serves the purpose of enhancing clarity for other users in discerning the source of messages and files they receive.
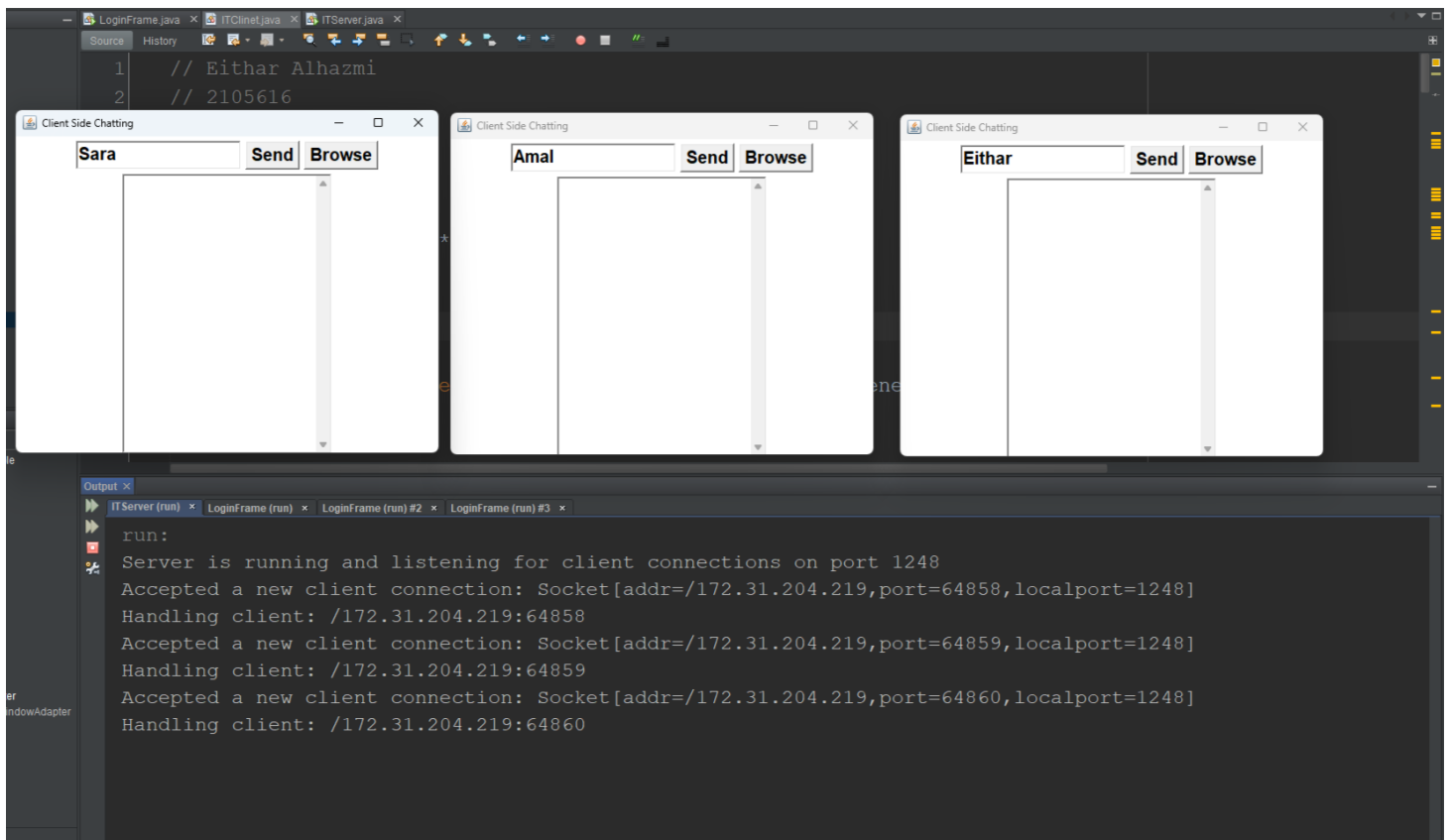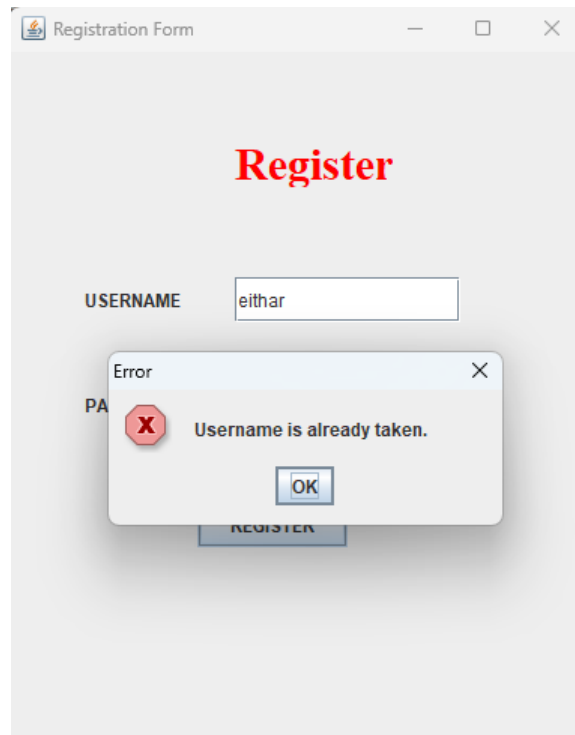
Provide a menu-based interface for the client to perform the following actions:
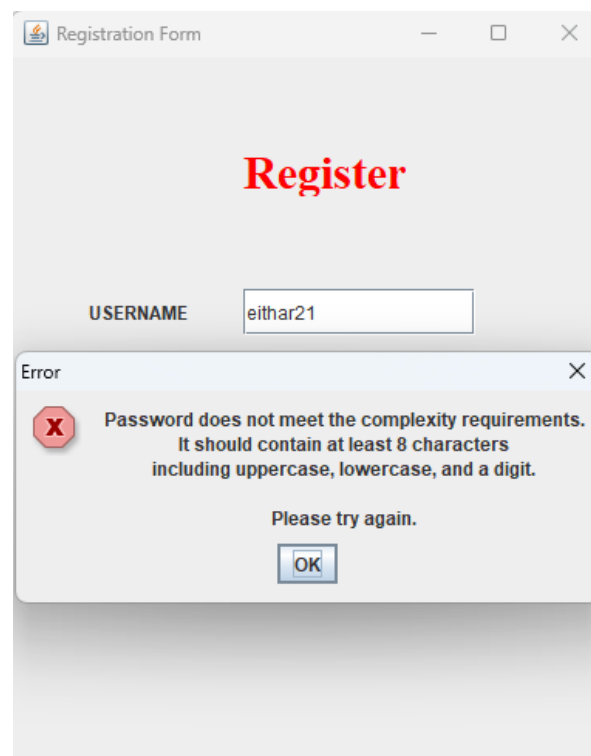
Send a text message to another client, Send a file to another client and View the received messages files and Logout from the application (in my application I use GUI so the user can logout by X button)
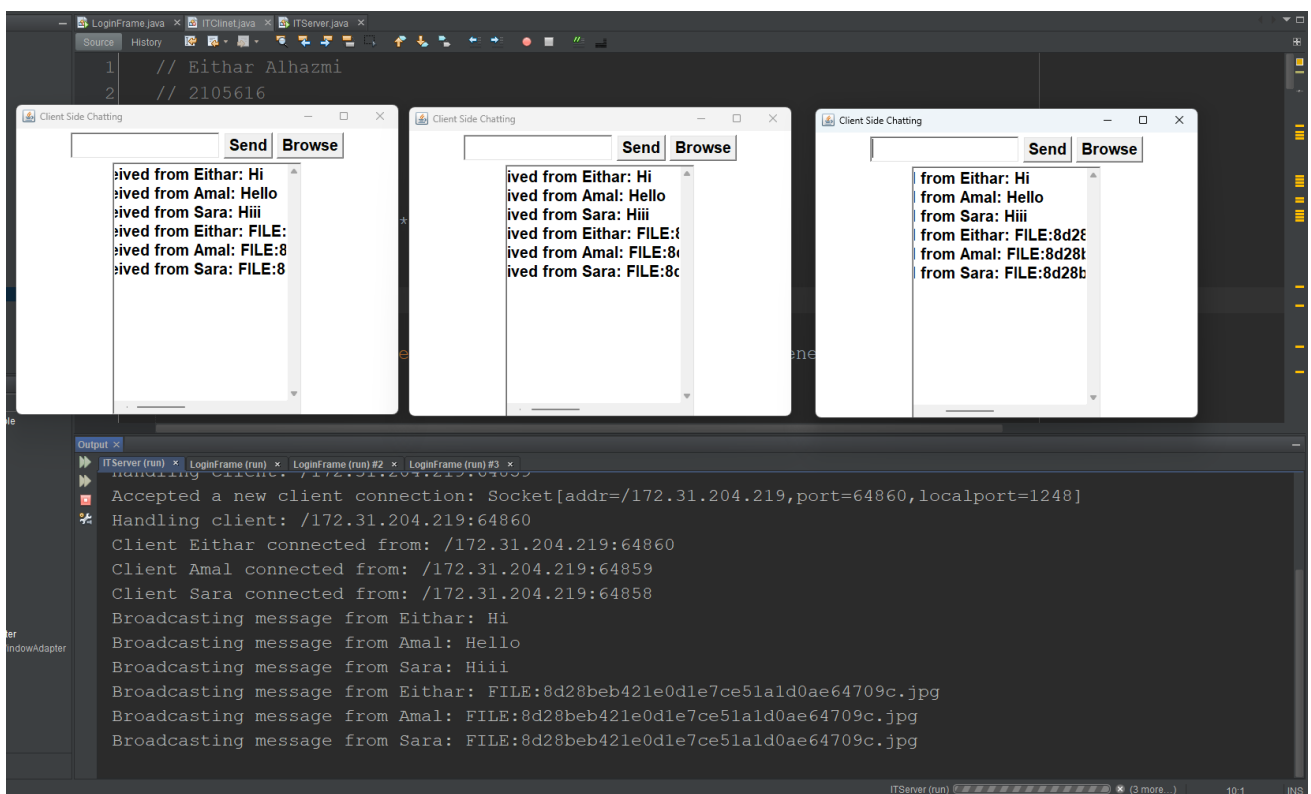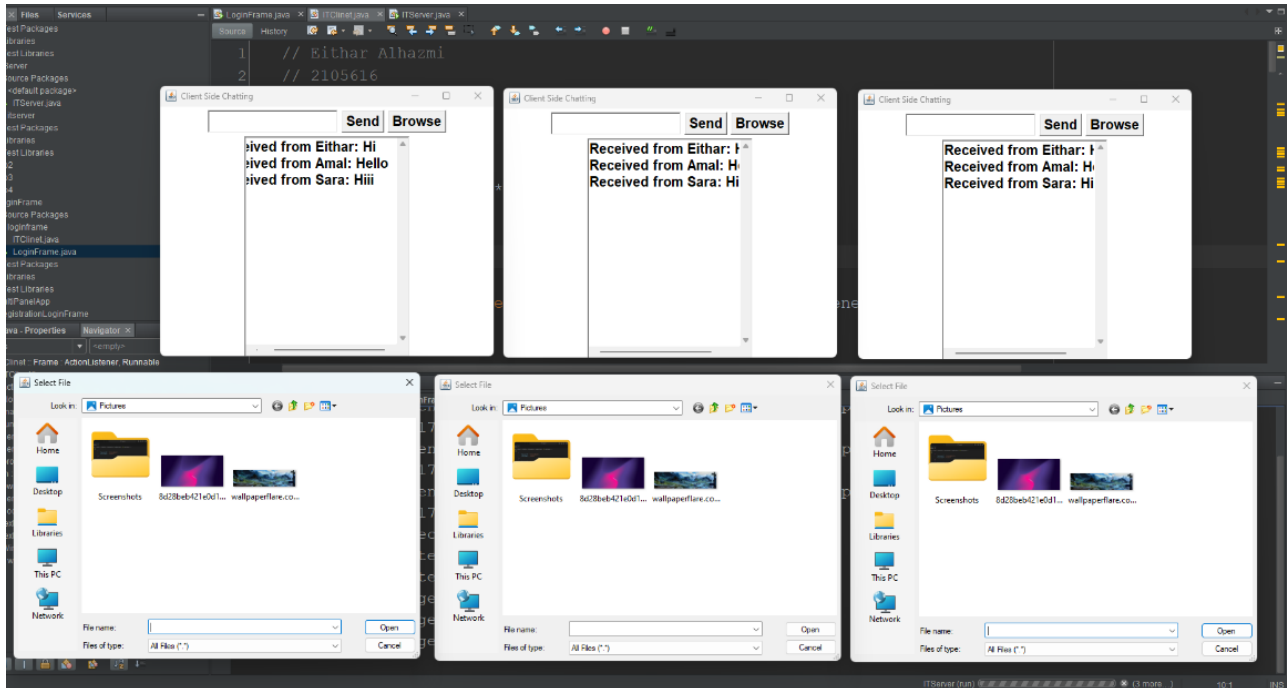
When the user try to register with username already booked up

when the user try to put password doesn't meet the complexity requirements

1.5 References

[1] "Java socket programming 4 - multi-client interactive sessions," YouTube, https://www.youtube.com/watch?v=ZIzoesrHHQo (accessed Nov. 1, 2023).

[2] "Java GUI login with Signup & signin (store and match username, password in file) - practical demo," YouTube, https://youtu.be/6eaMLjsodoc (accessed Nov. 1, 2023).

[3] "Login and register from JSON file in Java | Fun with java #3," YouTube, https://www.youtube.com/watch?v=1zkoa2X2NlM (accessed Nov. 1, 2023).

[4] "Chatgpt," ChatGPT, https://openai.com/chatgpt (accessed Nov. 1, 2023).

[5] M. Hameed et al., "Login page in Java Swing with source code tutorial - GUI," Tutorials Field, https://www.tutorialsfield.com/login-form-in-java-swing-with-source-code/ (accessed Nov. 1, 2023).