# CHATTER

Com S 486 – Term Project

Tyler McAnally

## Design Goals

Chatter is a desktop application that allows multiple users to sign in and chat within a group or with a specific person similar to Skype which was what this project was modeled after. The abstract goal/purpose of this application was not to fill a space to which was empty; its purpose was a personal one in which I was able to learn how to make several users communicate remotely and pass data around through a network and how encryption applies to an application.

Goal 1:

The first and most important goal was to send messages to a server and send that message to other users connected.

Goal 2:

With any messaging application, people are going to want to talk to specific people one-on-one.

Goal 3:

If a recipient is not logged on, they will get an email notification from the server letting them know a specific person is trying to reach them.

Goal 4:

Implement a simple user interface to make using the application easy.

**Uses**

- Businesses can quickly contact other coworkers.
- People can contact their clients for quick communication rather than emailing them.
- Friends or friend groups can instant message each other easily.

## System Architecture and Technology

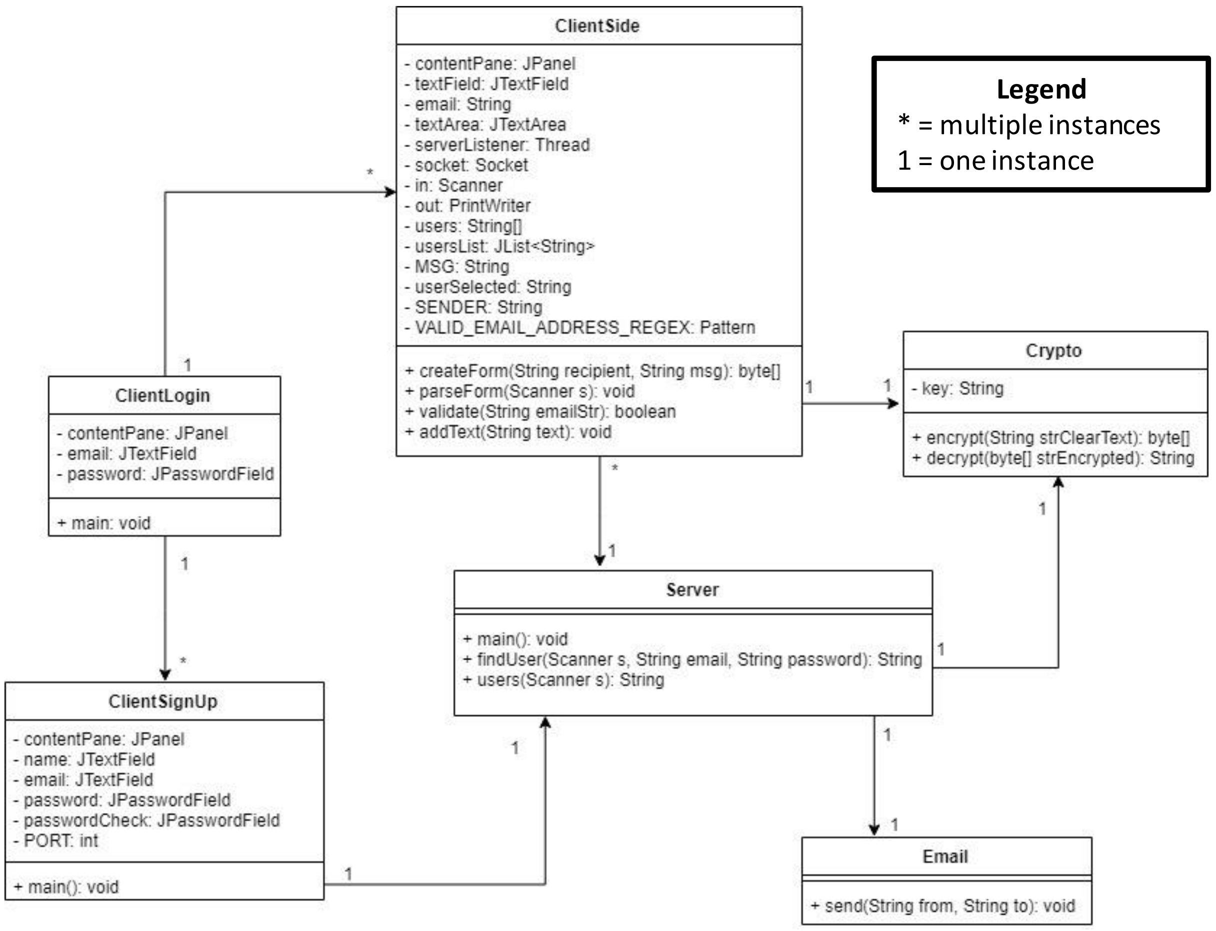
Chatter is a client-to-server application using Java 8. The Java classes Socket and ServerSocket were used to make a TCP connection that establishes a persisting client-server connection.

There are 6 classes that are used in this application:

- Server.java: This class is used for message transportation and user authentication. It saves user data in database.txt.
- ClientLogin.java: This class is used for logging in to the client if credentials match a user in the database. This class is also allows new users to bring up the ClientSignUp class in order to register.
- ClientSignUp.java: This class is used to register within the server so a user can go back to the ClientLogin window to sign in. This class only sends information to the server; it does not receive anything.
- ClientSide.java: This class makes the communication with the server. It extends thread so that it can run on its own. Here, a user can send a group message by default or a message to a specific person. There is a subwindow where messages show up from specific people, a list of users that are in the database, and a text field to send messages.
- Email.java: Email is used by the server to send email notifications to users that receive messages but are not online at the time the message is sent.
- Crypto.java: Crypto is a static class that holds the methods for encrypting and decrypting messages using the Blowfish algorithm.

In a further examination of the source code, you can see how the components work together. It is well-commented so it should be fairly easy to follow.

To get a better idea how this classes interact, here is the class diagram:



## Technologies

- Java 8
- Java Swing
- Email
- Blowfish cryptography
- Threads
- TCP Sockets
- Libraries: mail.jar (email), activation (email credentials)

## User Manual

Open the project in Eclipse. The zipped folder will have the testing database in it by the name “database.txt”. This is the database used by the Server. Make sure external libraries (mail.jar & activation.jar) are in the project or it may not compile or send email. Here are the links to download them if they are not available. Once downloaded, add jars to build path. I’ll include them in my zip file just in case. Just make sure they are part of the build path.

mail.jar

JavaMail: https://www.oracle.com/technetwork/java/index-138643.html

activation.jar

JAF: https://www.oracle.com/technetwork/articles/java/index-135046.html

How to run to test my goals:

Preparation: Make sure port 4444 is available on your system. If not, use another port that is.

1. Compile classes in package *core*.

2. Run Server. It should print to the console which port it is listening on.

3. Run ClientLogin and login to 3 separate users to test communication:

   Email: test1@gmail.com Password: test

   Email: test2@gmail.com Password: test

   Email: test3@gmail.com Password: test

   a) Using test1@gmail.com (it will say on the top of the window), type a message and send it. You will see whatever you typed show up in all windows. This is group chat and is used by default.

   b) Next, using the same window/user, click on a user in the side panel that is also logged in. This will clear your message area and when you send a message, it will show up on the current window and only the on the user’s window you clicked on. This is direct messaging. If you’d like, select another user and repeat (b) as long as they are logged in.

Warning: Once a user is clicked on, the user is locked out of group chat unfortunately. This is a known bug.

4. To test email communication, click the “Sign Up” button on the ClientLogin window. Register your valid email. Your password can be whatever you want. Once submitted, all windows will close. Restart the server so it can update (this is not ideal but this is the how the current state works). Start ClientLogin again and login to one of the test users listed above. You should see the new user on the right-hand side. Click on it and send a message. The server console should start sending the email. Login to the actual email you just made and check your inbox. You should have an email from chatterproject486@gmail.com.

Warning: A bug exists where the user that receives the email doesn’t get the message when they login.

If these steps are followed, the goals I stated will be fulfilled and will work. Not all edge cases, corner cases, and boundary cases were caught in development of this application so it can break if too many things are going on at once. If used properly according to the steps listed, there shouldn’t be any unexpected errors that occur.