Peer-to-peer Chat Project Report

by Erdi Gültekin

1.Project Summary

P2P Chat application is written with Java programming language and it aims to provide a peer-to-peer messaging solution.

2. Solution Approach

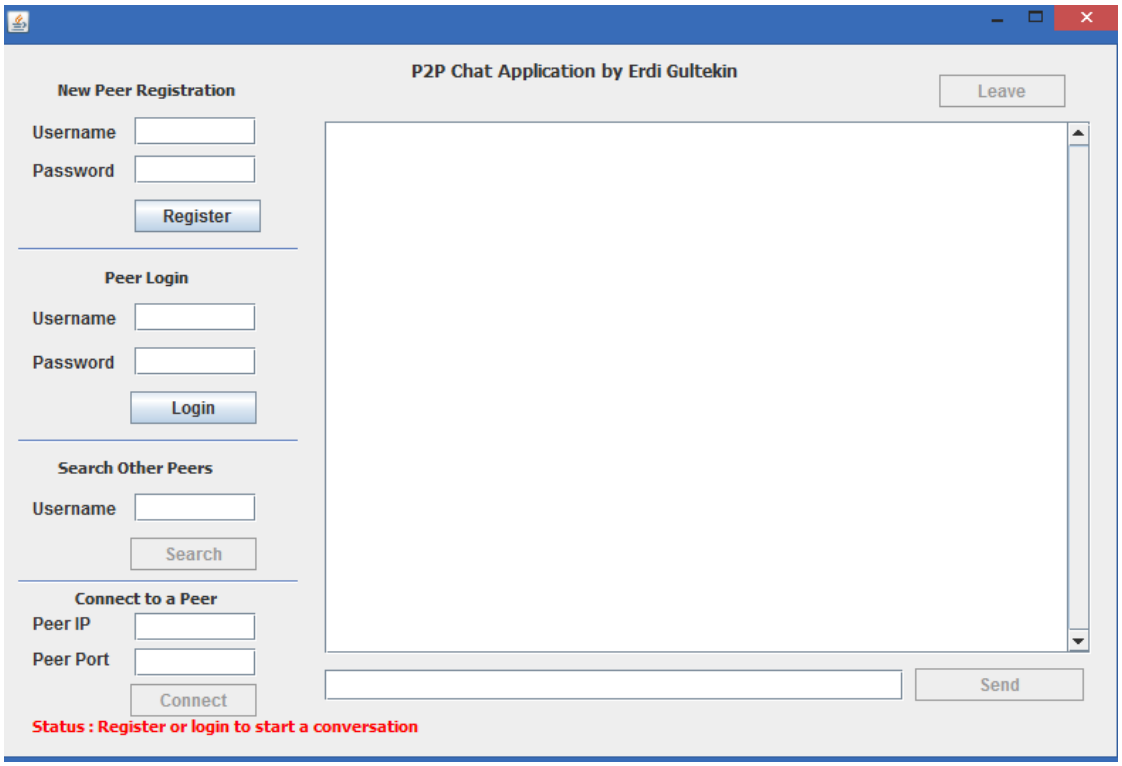
P2P Chat application consist of 2 parts. First is the peer program and the other one is the registrar program. Users are using the peer program to register their usernames, login to the system and search other users. After finding IP address and port number of any other user in the registration system, they can send request to them to start a chat.

Registrar program is responsible for keeping track of the registered users, their passwords, connection information (IP address and port) and online status. To make the testing easy with a single computer, it is assumed that both the registrar and the peer will be run on the “localhost” and it can be changed from the peer and registrar.

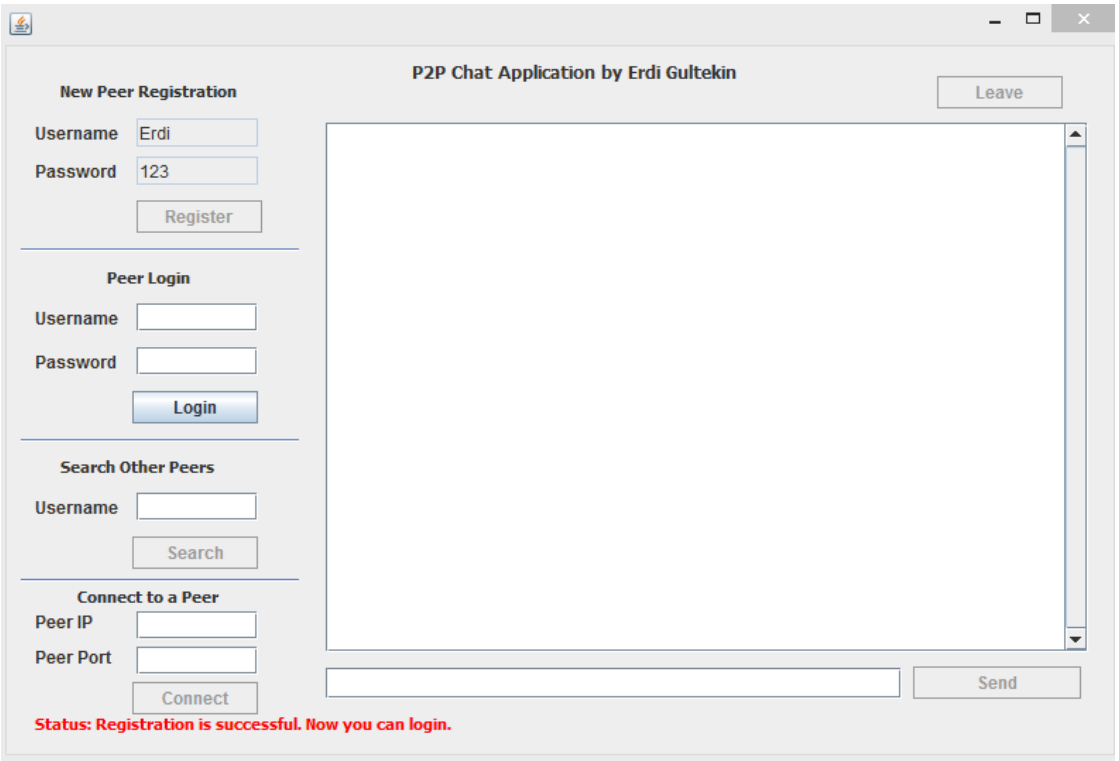
3. Usage Explanation

Usage of P2P Chat program is fairly simple. Firstly, we run the **Registrar** program to keep track of peers.

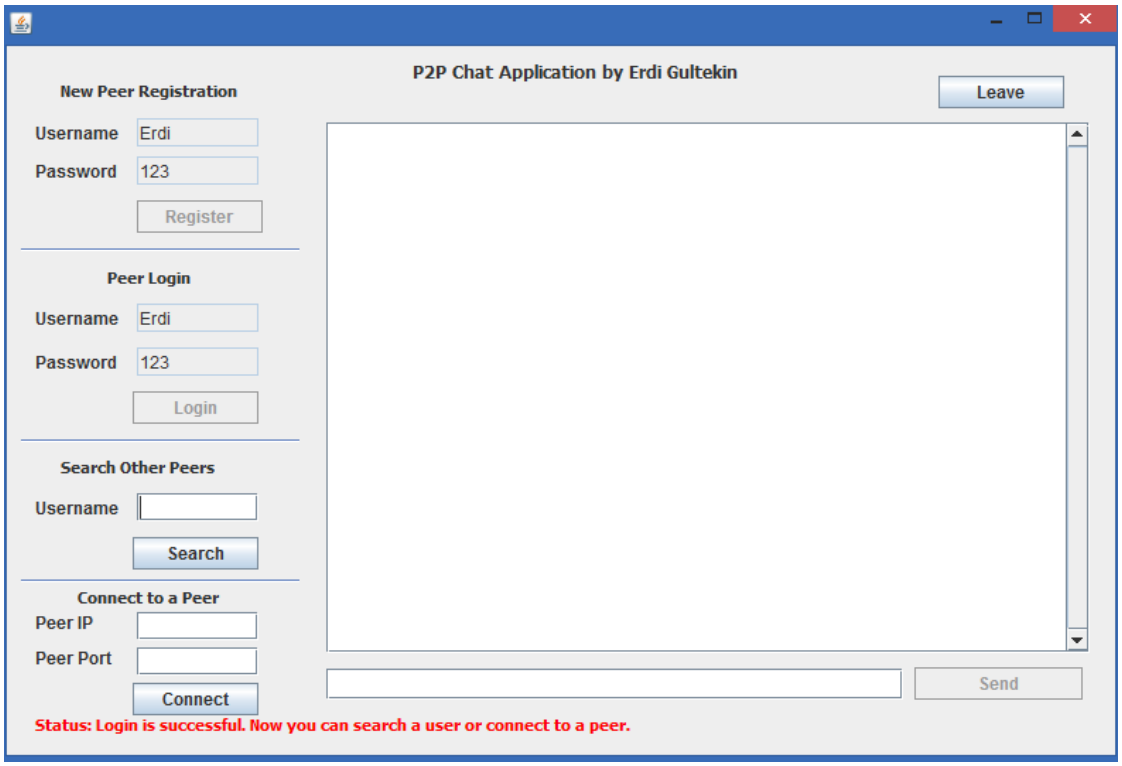
Then we can run the **Peer** program and the screen below will be opened:



In this screen, we have 2 options. We can register a new username with password or alternatively, we can login with an registered username and password.

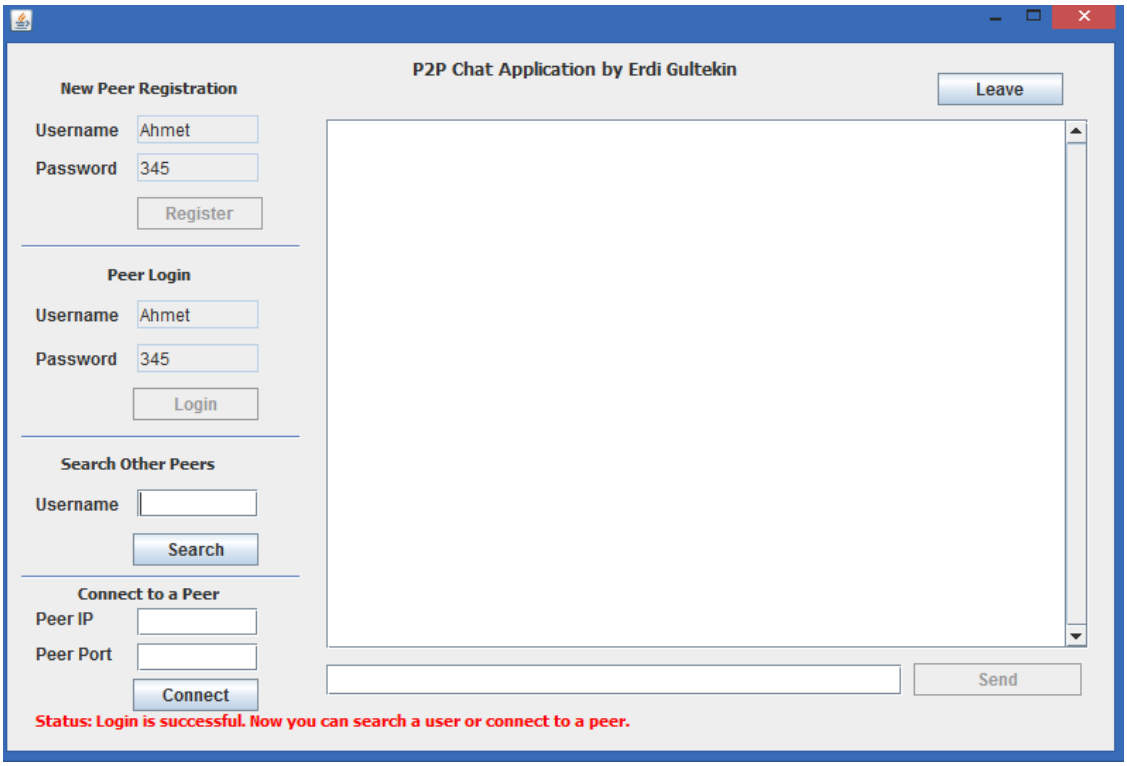


After the registration, we will be ready to login as in the screenshot above. We can now enter our username and password to login to the system.

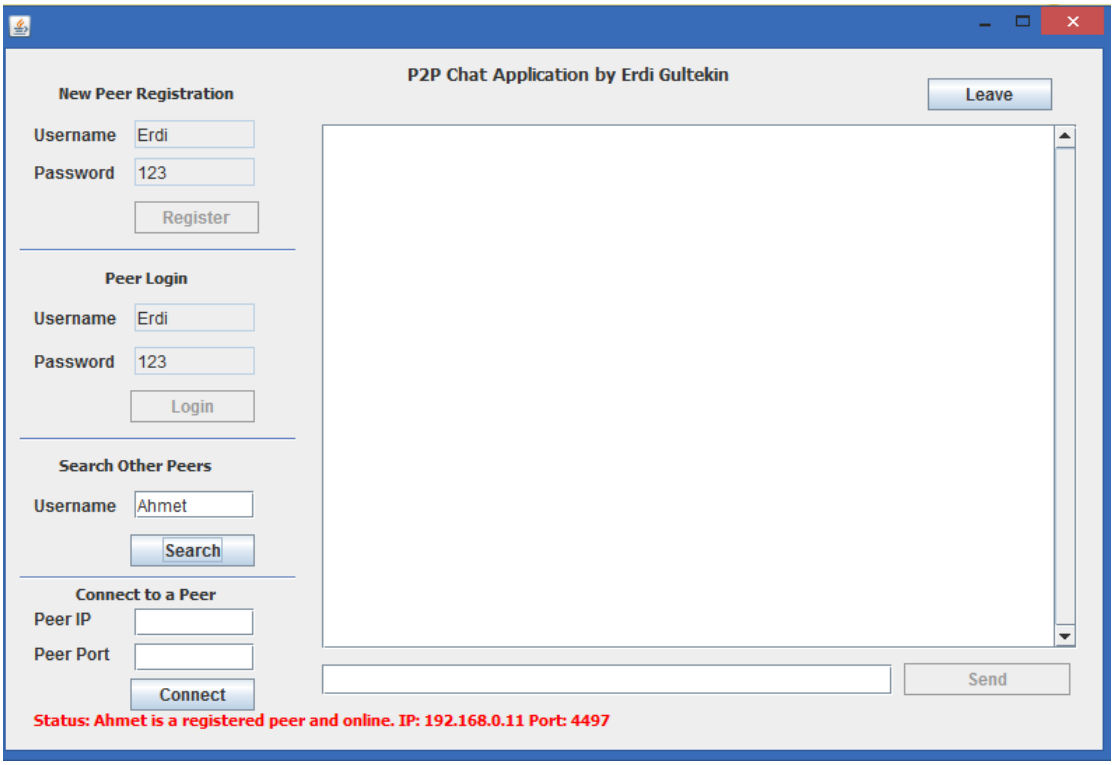


After the login successful message, we can now search for other peers or fill the connection information of another peer if we already know them.

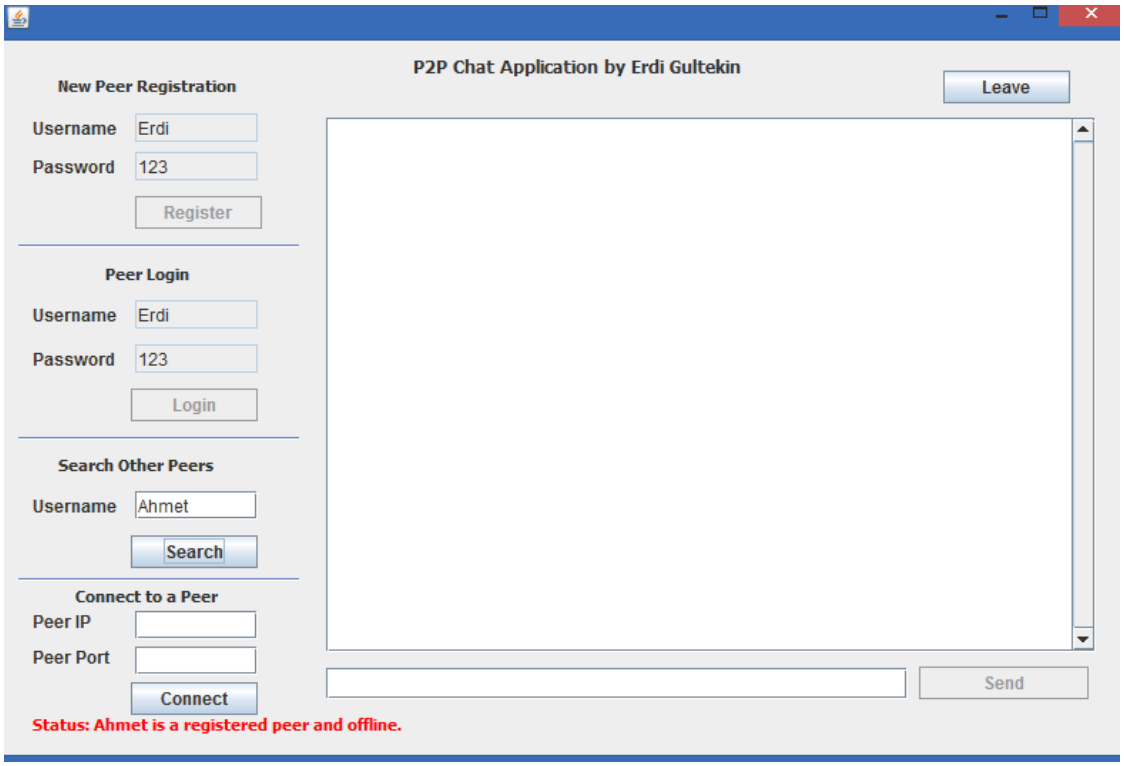
Let's start another instance of the Peer program and do the same steps above with a different username (e.g. register a new user with "Ahmet" name) to test the system:



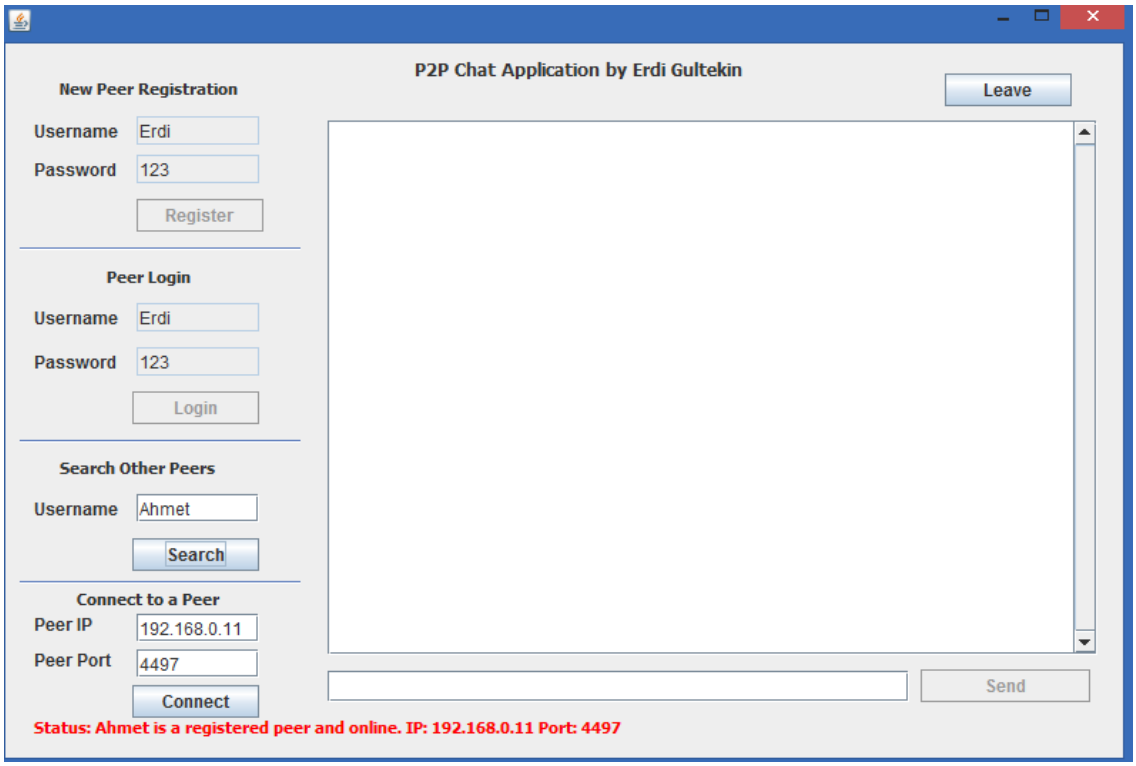
In order to connect to another user (e.g. Ahmet) for chatting, we need to get its connection details. To get this information, type the username of the peer (e.g. Ahmet's name) into the search box and click on the Search button.



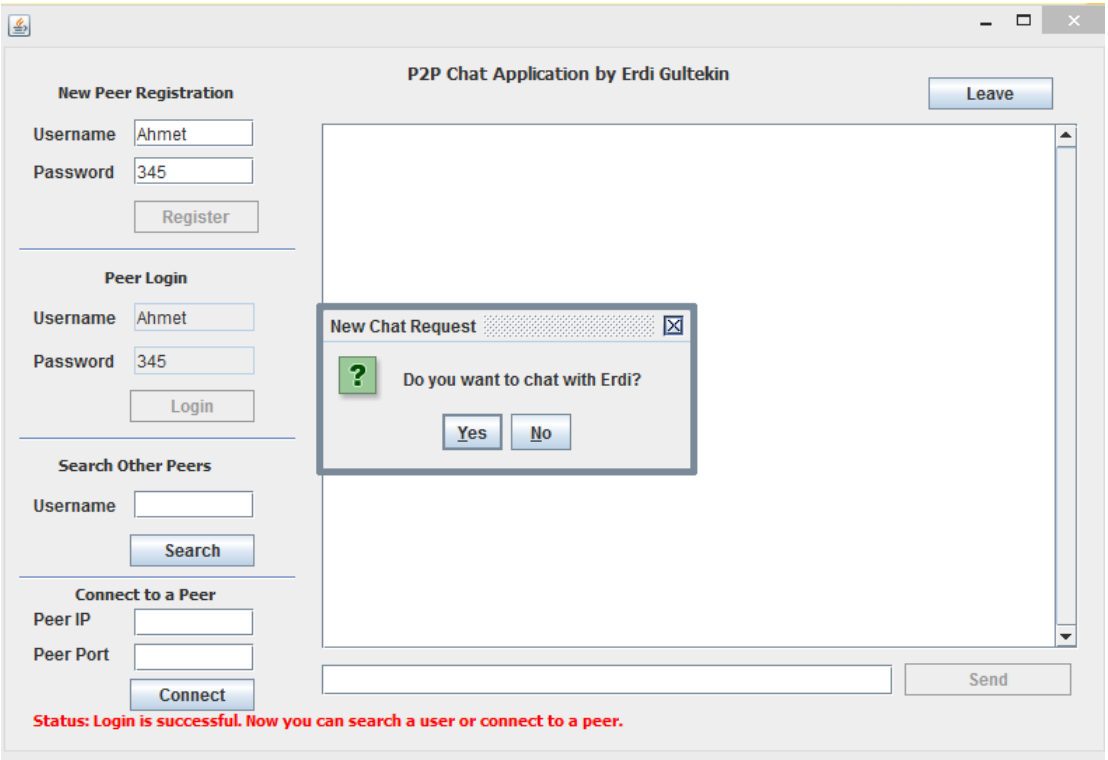
The registrar will return us the connection details of other peers and their online/offline status (e.g. whether Ahmet is a registered peer and online and his IP address and port number if he is online). If the user was offline, we would get the following message:



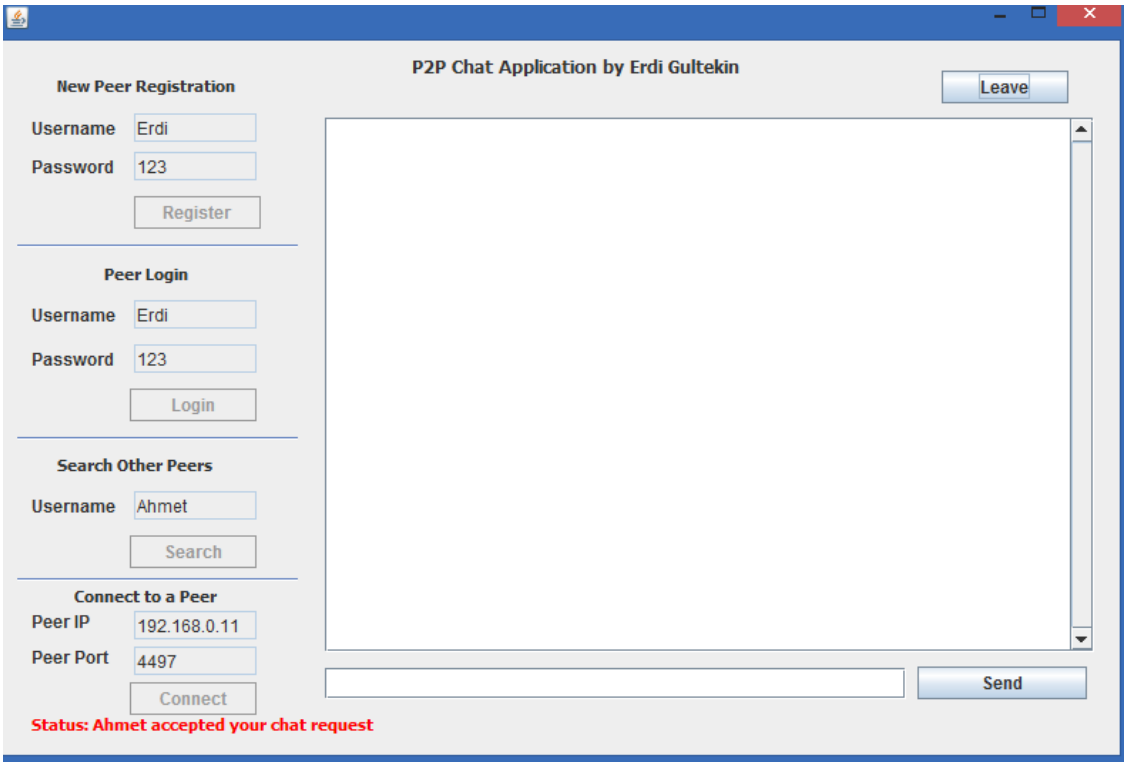
Fill the connection details and click to Connect button:

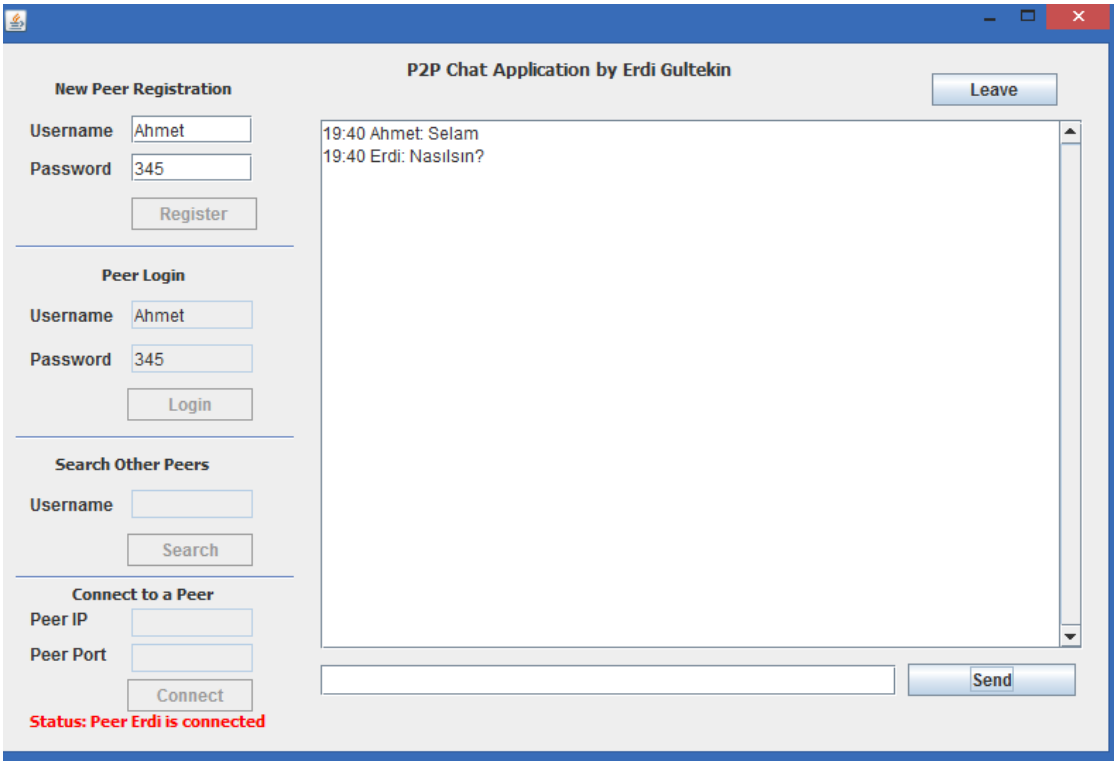


The other peer will get a notification about your chat request. He can accept or reject your chat request:



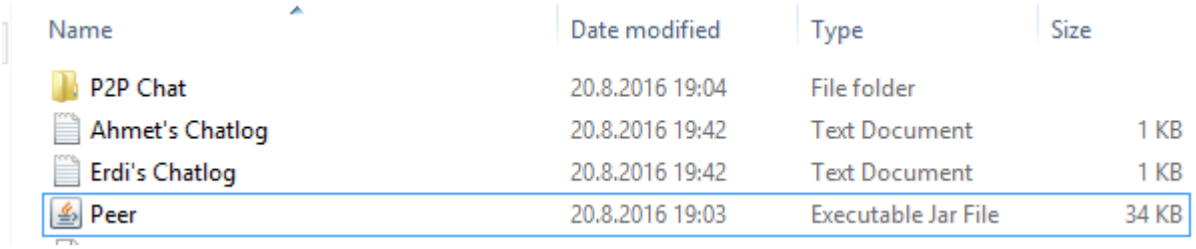
After the confirmation of the other peer, now the users can send and receive messages from each other:



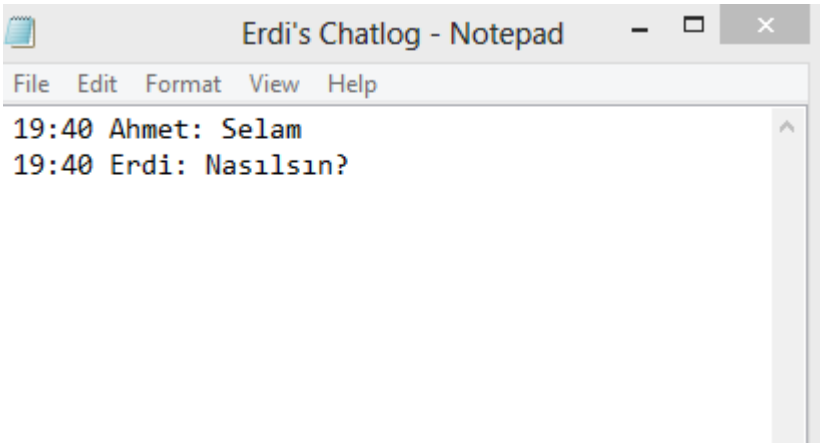


Users can leave the system by either clicking on the Leave button or the close button (X) at the top right corner of the screen.

After leaving the system, all chat message logs will be recorded to the same directory:



Name	Date modified	Type	Size
P2P Chat	20.8.2016 19:04	File folder	
Ahmet's Chatlog	20.8.2016 19:42	Text Document	1 KB
Erdi's Chatlog	20.8.2016 19:42	Text Document	1 KB
Peer	20.8.2016 19:03	Executable Jar File	34 KB



4. Protocols

The registrar is using the java.net.ServerSocket and java.net.Socket classes (like a server) to connect with the Peer program. The registrar handles each new peer connection by creating a new thread. Peer program is using only the java.net.Socket class (like a client) to connect with the registrar. These classes are using the TCP (Transmission Control Protocol) which is a reliable and stream oriented protocol. It is assumed that the registrar and peers will run on the localhost and use the predefined port 5555.

Messages sent between the registrar and peers with a Packet class instance. This class provides a basic communication structure and it allows to transfer peer registration, login and leave messages.

Peers have separate communication channels between themselves. They use java.net.ServerSocket and java.net.Socket classes to act like a server and a client at the same time to send and receive message. Their communication is again based on TCP. Their IP addresses can be different and their communication ports are assigned randomly by the registrar. This makes it easy to have separate channels of communication and run them on the same computer for testing purposes. Peers have a Peer class, a GUI (graphical user interface) class and a PeerThread class. PeerThread class is handling the server like processes in the peers and Peer class is handling the client like processes.

Messages between the peers are sent via different packet which is called ChatPacket class. This class provides communication with the message types like chat requests, chat messages, message delivery status, peer online/busy/offline status and other needed connection information.

The overall communication structure between the peers and registrar is depicted as below: