Introduction

In this project, you are going to develop a social networking application implementing client and server modules. (i) The *Server* module manages message transfers, notifications and friendship relationships among the users, and (ii) the *Client* module behaves as a user which adds/removes friends, accepts/rejects friendship requests, sends/receives messages and receives relevant notifications.
The server listens on a predefined port and accepts incoming client connections. There might be one or more clients connected to the server at the same time. Each client knows the IP address and the listening port of the server (to be entered through the Graphical User Interface (GUI)). Clients connect to the server on a corresponding port and identify themselves with their names. Server needs to keep the names of currently connected clients in order to avoid the same name to be connected more than once at a given time to the server.
On the server side, there is a predefined database of users which are presumed to be registered to the social network so that you do not need to implement any registration process between a client and the server. That user database has actually been provided you together with this document. A client, whose name should be in the user database, will be able to connect by providing his/her name only (no password or other type of security). Once connected, he/she will act as mentioned in the rest of this document.
The abovementioned introduction is for the entire project, which is to be completed in three steps. Each step is built on the previous step and each has specific deadlines and demos.

Project Step 1

After the server starts listening, clients start to connect to the server. A connected client can broadcast textual messages to all other connected clients via the server. In other words, server behaves like a bridge among all clients. When a particular user sends a message, server forwards that message to other users, except the sender. Therefore, server needs to keep connected clients' list up-to-date.

Users perform all of the operations through a GUI; such as connecting to the server, entering their name, sending message, etc. Additionally, all received messages with the sender's name must be shown on the client GUI.
This step is the basis of the project. Although the project has some friendship management and notification features, you do not have to implement those in this step of the project. You will implement them in coming steps.
For programming rules and submission specifications, please read the corresponding sections at the end of this document.
Server Specifications:
 There is only one server running on this system.

 The port number on which the server listens is not to be hardcoded; it should be taken from the Server GUI.

 The server will start listening on the specified port. It has to handle multiple clients simultaneously. To do so, whenever a client is connected to the listening port, the corresponding socket should be added to a list and the server should continuously accept other client sockets while listening.

 Only users which reside in the user database can connect to the server as a client. In other words, no user with a name that does not exist in the user database can connect.

 The server forwards the incoming messages to all clients other than the sender, but the server should include the name of the sender to the forwarded messages.

 All activities of the server should be reported using a rich text box on the Server GUI including the names of connected clients as well as all the message transfer details. We cannot grade your project if we cannot follow what is going on; so the details contained in this text box is very important.

 Server must handle multiple connections. At the same time, one or more clients can send and receive messages to/from the server.

 Connected clients' names must be unique; therefore, the server must keep track of connected clients' names. If a new client comes with an existing name, server must not accept this new client.

 When the server application is closed (even abruptly), nothing should crash! Also, the process in the operating system regarding the server should be terminated properly.

Client specifications:
 The server's IP address and the port number must not be hardcoded and must be entered via the client GUI.

 There could be any number of clients in the system.

 If the server or the client application closes, the other party should understand disconnection and act accordingly. Your program must not crash!

 All activities of the client should be reported using a rich text box on the client GUI including sent and received message information. We cannot grade your project if we cannot follow what is going on, so the details contained in this text box is very important.

 Each client must have a unique name. This name must be entered using the client GUI. This name is also sent to the server. The server identifies the clients using their names.

 Each client can send and receive textual messages at any time. If a client sends a message, this message is forwarded to all other online (connected) clients by the server.

 Each client can disconnect from the system at any time. Disconnection can be done by pressing a disconnect button on client GUI or by just closing the client window.

 If the client application is closed (even abruptly), nothing should crash! Also, the process in the operating system regarding the client should be terminated properly.

 Both connection and message transfer operations will be performed using TCP sockets.

Project Step 2 Second step of the project is built on top of the first step. In this step, you will modify previous client and server modules to add more functionalities.

In this step, in addition to the message transfer feature of step 1, adding friends and notification features are to be added to the application.

Each user can add friends selected from the user database. This procedure works with an invitation mechanism. The inviting user (**inviter**) should send an invitation through the **server** for asking the consent of the user (**invitee**) who has been invited to be a friend of the inviter. Each invitation and corresponding acceptance or rejection decision should be relayed by the server between the inviter and the invitee. In this sense, the invitee should have a choice to accept or reject a friendship invitation and this decision should be made via the GUI of the invitee. The design of this GUI part where the client makes the decision of accepting/rejecting the friendship request is up to you, but we expect each project group to find a good, functional and user friendly engineering solution here. Moreover, please notice that there are some other requirements of invitation handling as will be mentioned below paragraphs.

There are also some important issues related to friendship invitation handling. In the rest of this document, connected clients (to the server) are referred as **online** users, while the others (unconnected clients) are referred as **offline** users.

i. The invitee who receives an invitation from an inviter may be offline or online. In such cases, the invitee should receive the invitation as soon as he/she is online. For example, if the invitee is already online at the time the inviter sends the invitation, the invitee gets the invitation. However, if the invitee is offline at the time the inviter sends the invitation, the invitation should be in a pending state during the period that the invitee is offline, and the server should relay the invitation as soon as the invitee becomes online.
ii. There might be multiple invitations (by different inviters), which are directed to a particular invitee. The invitee can respond (accept/reject) to these invitations at any time. That means, there is **no rule** such that the invitee cannot perform anything else when there is a pending invitation. All pending invitations need to be kept in a data structure so that the invitee can respond any time that he/she wants.
iii. The invitee should be able to send an invitation or a message while it has pending invitations. Your program should also handle such cases using threads.

Another mechanism that should be implemented in this step is the *notification*. Whenever an invitation is responded by the invitee with accept or reject, this decision must be forwarded by the server to the corresponding inviter. If this inviter is offline at that time, the notification should be sent by the server as soon as the inviter goes online. This notification should be reported using the rich text box on the client and the server GUIs.
If there is a pending invitation between two users, there should not be another one between them until the pending invitation is responded. This needs to be handled by the server and related notifications should be sent to the inviter (consider online and offline cases).
The server should also keep track of the friendship relations. There must be a button at the client GUI to ask for current friends of that client from the server. In response, the server should send the list of current friends of that client to be displayed on the client GUI.
As in the step 1, all of the operations must be clearly shown on the client and server GUIs.
For programming rules and submission specifications, please read the corresponding sections at the end of this document.

Project Step 3
Third step of the project is built on top of the first and second steps. In this step of the project, remaining friendship relations are to be implemented in the application, which are (i) Removal of an existing friend and (ii) sending message to the existing friends.
A client can remove his/her friend without asking for any consent from him/her. However, removed friend should be notified. Again, the server is in the loop and same online/offline cases apply.

A client should also be able to send messages to all of his/her current friends through the server. The online friends should receive the message immediately. The offline friends should receive the message as soon as they become online. As in the step 1 and 2, all of the operations must be clearly shown on the client and server GUIs.