

12/01/2020

C# Project Chat App

Functions implemented:

- Create profile
- Login
- List topics
- Create topics
- Send messages to everyone

Functions not finished:

- Send messages to people in a specific topic

Functions not implemented:

- Send a private message

Possible improvements:

- Adding more operations to the database of user profiles (list profiles on the server interface, update username/password, delete)

Design

The application rests on a few structures.

```
enum MsgType
{
    createprofile = 0,
    login = 1,
    listtopics = 2,
    createtopic = 3,
    listusers = 4,
    sendmsg = 5,
    sendprivmsg = 6,
    switchtopic = 7,
    help = 8
}

struct Message
{
    public MsgType Mymsgtype { get; set; }
    public List<string> S { get; set; }
    public string Topic { get; set; }

    public Message(MsgType msgtype, List<string> listofString, string tpc = null)
    {
        this.Mymsgtype = msgtype;
        this.S = new List<string>(listofString);
        this.Topic = tpc;
    }
}

struct Client
{
    public TcpClient s;
    public string username;
    public string topic;

    public Client(TcpClient client, string usn, string tpc)
    {
        this.s = client;
        this.username = usn;
        this.topic = tpc;
    }
};
```

These are instantiated in the server as such:

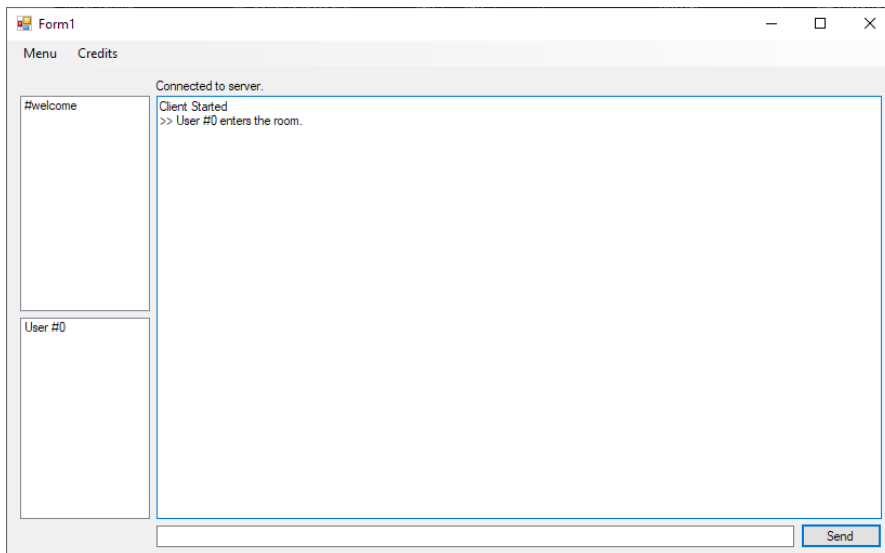
```
List<Client> clients = new List<Client>();
List<Profile> profiles = new List<Profile>();
Dictionary<string, List<Client>> topics = new Dictionary<string, List<Client>>>();
```

Then, the server runs by waiting a TCP connection. Once it receives a connection, it creates a client associated to it and runs a thread dedicated to this client. This thread will then wait for any new instruction to execute by the server. When a client is closed, the exception is caught and the client is disposed of, as well from the list of clients.

On the client side, when the application is starting, it will try to connect to the server. On success, it will also create a subsidiary thread to listen to any instruction sent by the server. The instructions provided by the client are relayed to the server and then back to the same client, as well as the other clients if the instruction was a message to everyone, for example.

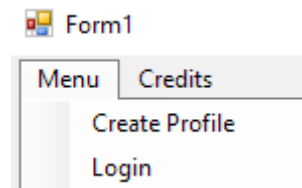
Screenshots

Client connection

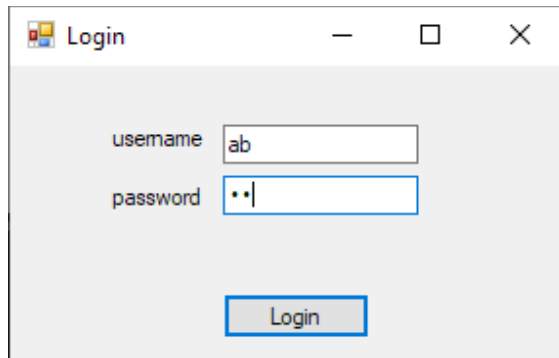


Login

From the toolbar, we can access the create profile and login.

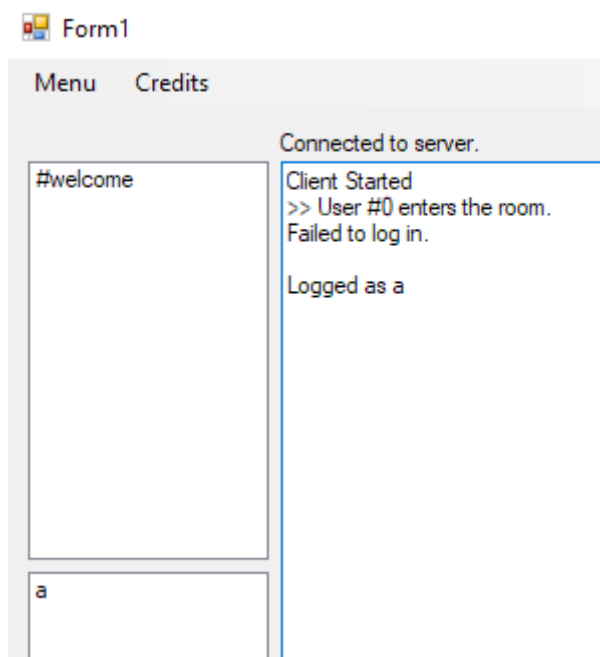


We try to log in as a user that does not exist in the database.

A screenshot of a 'Login' dialog box. It has a title bar with a standard Windows icon and window controls. Inside, there are two input fields: 'username' with the text 'ab' and 'password' with two dots. Below the fields is a 'Login' button.

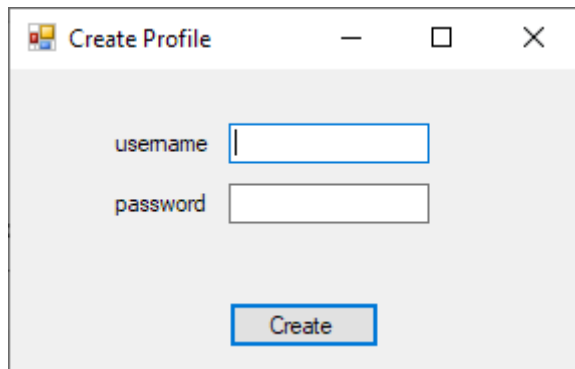
```
Connected to server.  
Client Started  
>> User #0 enters the room.  
Failed to log in.
```

If the user exists and the password corresponds, we are logged as the user “a”. The listbox of users is updated as well.

A screenshot of the 'Form1' interface after a successful login. The 'Menu' tab is active. A text area on the right displays the following text: 'Connected to server.', 'Client Started', '>> User #0 enters the room.', 'Failed to log in.', and 'Logged as a'. Below this text area is a listbox containing two items: '#welcome' and 'a'.

Create a profile

The layout is the exact same as the one for logging in. It is needed to create at least one profile to get the database up and thus doing the operations on this database. The database is in SQLite, thus if already installed, the database can be created on the spot.



A screenshot of a 'Create Profile' dialog box. The window has a title bar with the text 'Create Profile' and standard minimize, maximize, and close buttons. The main area is light gray and contains two text input fields. The first field is labeled 'username' and the second is labeled 'password'. Below these fields is a 'Create' button. The 'username' field has a blue border and a cursor, while the 'password' field has a gray border. The 'Create' button also has a blue border.

Field	Label
username	username
password	password

Create

Multiple users

As depicted below, the application can handle multiple clients sending to everyone messages.

The image shows three instances of a chat application window titled 'Form1'. Each window has a 'Menu' and 'Credits' section at the top. The main area is divided into two panes: the left pane shows a list of users ('User #0', 'User #1', 'User #2'), and the right pane shows a log of messages. The messages include 'Client Started', 'User #0 enters the room.', 'User #1 enters the room.', 'User #0: First message', 'User #1: Second message', 'User #2 enters the room.', and 'User #2: Third message'. The timestamps for the messages are [06:00:19], [06:00:26], and [06:00:39].

Leaving is also a message sent to everyone. With the command `"/createtopic"`, we can add a topic to the listbox of topics to everyone. However the development of the messages specifically for a channel is not over.

The image shows three instances of the chat application window titled 'Form1'. The first two windows show the same message log as the previous image, but the third window shows a new message: 'User #2 left the server.' and 'User #2 enters the room.' The 'Menu' section now includes a new topic, '#new_topic'. The 'Credits' section shows the command `/createtopic new_topic` entered in the input field.