

Relatório

“Chatup.NET”

.NET Remoting - simple Internet relay chat application

Grupo 10

Carlos Manuel Carvalho Boavista Samouco

up201305187@fe.up.pt

Diogo Belarmino Coelho Marques

up201305642@fe.up.pt

José Alexandre Barreira Santos Teixeira

up201303930@fe.up.pt

2 de Abril de 2017

Índice

| | |
|----------------------------------|----|
| 1 Aplicação | 2 |
| 2 Arquitetura | 7 |
| 3 Serviços | 9 |
| 3.1 <i>SessionService</i> | 9 |
| 3.2 <i>LobbyService</i> | 11 |
| 3.3 <i>RoomService</i> | 12 |
| 3.4 <i>MessageService</i> | 14 |
| 4 Diagramas UML | 16 |
| 4.1 <i>ChatupClient</i> | 17 |
| 4.2 <i>ChatupCommon</i> | 18 |
| 4.3 <i>ChatupServer</i> | 19 |
| 5 Conclusão | 20 |
| 6 Recursos | 21 |
| 6.1 Referências bibliográficas | 21 |
| 6.2 <i>Software</i> utilizado | 21 |

Resumo

Este trabalho foi desenvolvido no âmbito da unidade curricular *Tecnologias de Distribuição e Integração* do Mestrado Integrado em Engenharia Informática e Computação (MIEIC) da Faculdade de Engenharia da Universidade do Porto (FEUP), e pretendeu-se desenvolver um sistema de IRC (*internet relay chat*) baseado em *.NET Remoting* na linguagem C# com interface gráfica (GUI) nos clientes utilizando Windows Forms. Cada utilizador está previamente registado num servidor comum, sendo conhecido o seu *username* único (uma única palavra), nome real e uma *palavra-passe*. Cada utilizador usa uma aplicação cliente que deverá ser a mesma para todos os utilizadores.

1 Aplicação

A primeira operação a efetuar pelo utilizador ao executar a aplicação cliente é realizar a sua autenticação através da combinação *username/password* atribuída. Os utilizadores que ainda não possuam conta podem eventualmente registar-se, carregando na opção “Register”. Por outro lado, é possível configurar a localização do servidor central (endereço IP, porto) carregando na opção “Configure” presente na mesma janela. Depois de introduzir os seus dados, o utilizador deve carregar em “Validate” para os validar junto do servidor, prosseguindo com a autenticação.

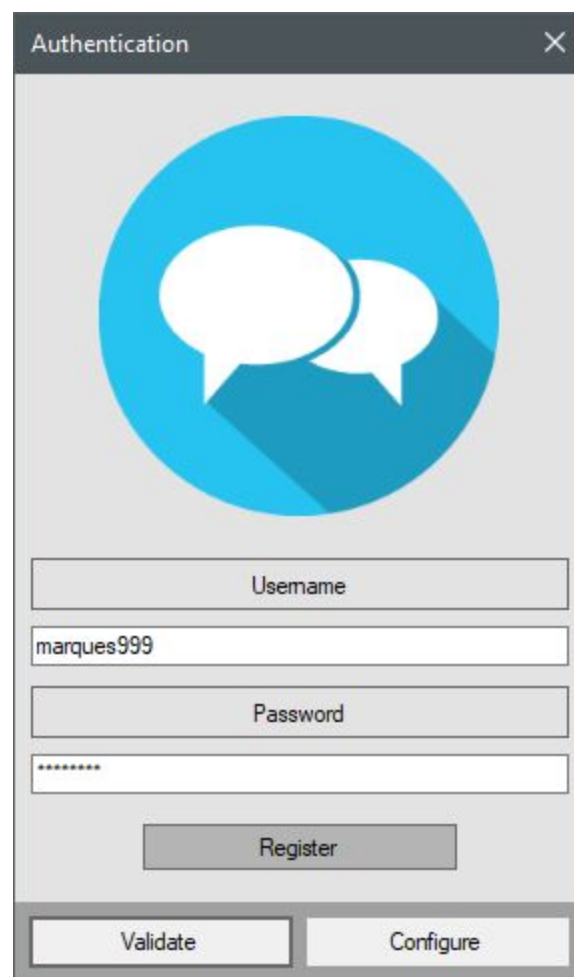
The image shows a window titled "Authentication" with a close button (X) in the top right corner. The window has a light gray background. In the center, there is a large blue circle containing two white speech bubbles. Below this graphic, there are three input fields: the first is labeled "Username" and contains the text "marques999"; the second is labeled "Password" and contains seven asterisks "*****". Below the password field is a "Register" button. At the bottom of the window, there are two buttons: "Validate" on the left and "Configure" on the right.

Figura 1. Janela de login da aplicação cliente, onde é possível registar uma nova conta de utilizador, autenticar-se no servidor, bem como configurar a localização do servidor central.

Caso a autenticação do utilizador seja bem sucedida, a aplicação pede ao servidor uma lista com todos os utilizadores registados no sistema, tanto dos que estão no estado ativo (com *login* efetuado), como dos utilizadores inativos (sem *login* efetuado). Os utilizadores ativos surgem na *interface* com um endereço na forma *tcp://hostname:port/messaging.rem* ao lado do respetivo *username*, enquanto os utilizadores inativos aparecem simplesmente como “Offline”.

Recorrendo a eventos (uma das principais funcionalidades da linguagem C#), a lista de utilizadores mantém-se atualizada em tempo real na *interface* da aplicação cliente, ou seja, sempre que surge um novo utilizador (registo) ou algum utilizador inicia sessão, os outros conseguem vê-lo imediatamente. A mesma coisa acontece quando um utilizador executa um *logout* ou simplesmente fecha a janela do cliente.

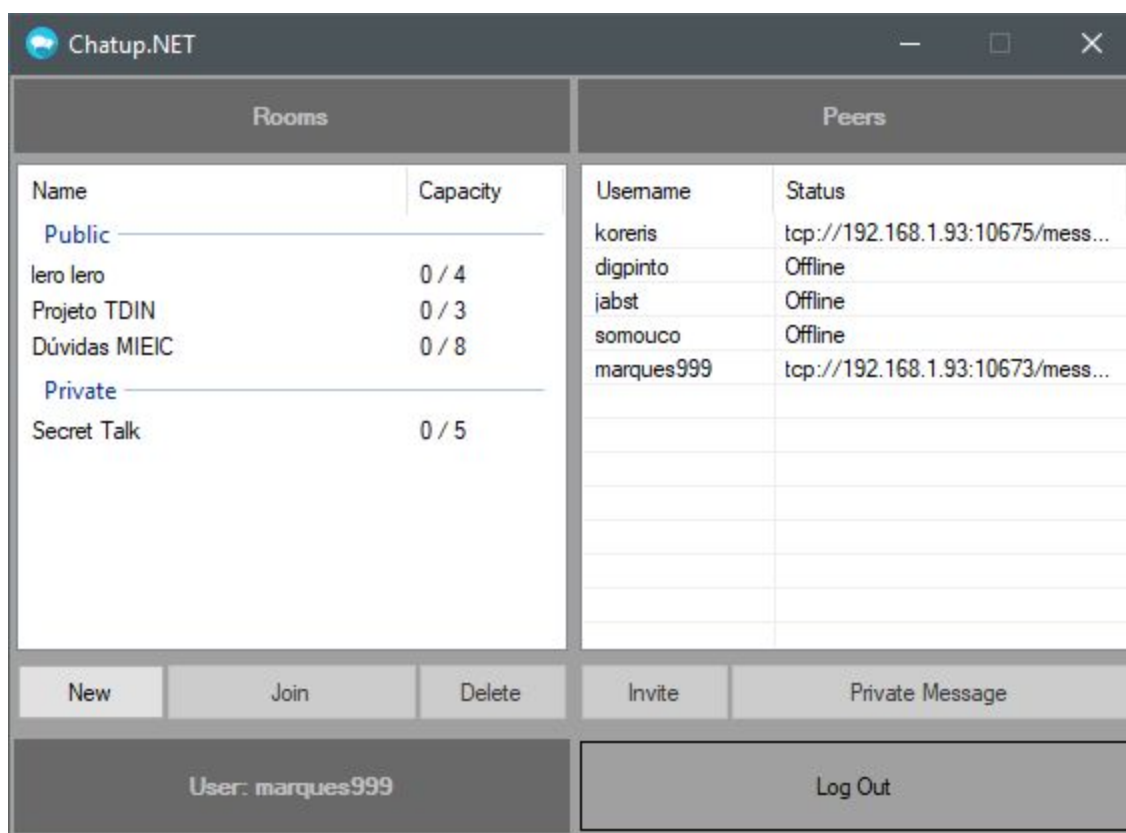


Figura 2. Janela inicial da aplicação cliente, evidenciando a lista de salas, a lista de peers e as diferentes ações que podem ser realizadas (New, Join, Delete, Invite, Private Message, Logout)

Existe ainda na *interface* da aplicação cliente uma lista com todas as salas de *chat* criadas pelos utilizadores, bem como informações relacionadas com a lotação e capacidade. Semelhante ao que se verifica com a lista de utilizadores, a lista de salas mantém-se atualizada em tempo real, ou seja, sempre que surge uma nova sala ou uma sala é apagada pelo respetivo proprietário, os outros clientes conseguem ver essas alterações imediatamente. A mesma coisa acontece quando um utilizador entra numa sala ou abandona essa sala, sendo que neste caso a contagem da lotação é atualizada (aumentando ou diminuindo). É possível escolher entre um dos utilizadores ativos para solicitar uma conversa privada, criar uma sala de *chat* para conversar em grupo com múltiplos utilizadores, ou participar numa das conversas de grupo existentes. Ao solicitar uma conversa privada, caso esta seja aceite pelo outro utilizador, é lançada uma nova janela de conversa em ambos os lados da comunicação. No caso do cliente escolher participar numa sala de *chat*, este deve ter em atenção ao tipo da mesma, pois existem salas públicas, nas quais qualquer utilizador pode entrar, no entanto a aplicação permite também que existam salas privadas, protegidas com uma *password* definida no momento da sua criação. Os clientes podem entrar numa sala privada de duas formas: introduzindo a *password*, se esta for conhecida, ou através de um convite, que pode enviado por qualquer um dos utilizadores que já se encontrem presentes na sala.

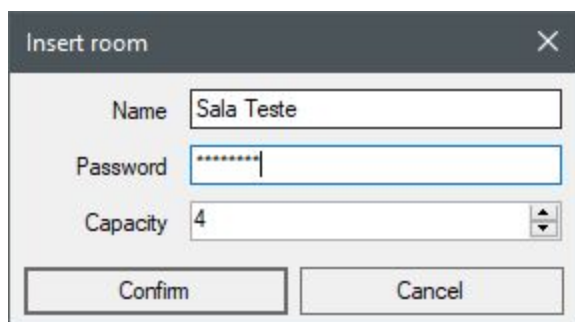
A screenshot of a dialog box titled "Insert room" with a close button (X) in the top right corner. It contains three input fields: "Name" with the text "Sala Teste", "Password" with masked characters "*****", and "Capacity" with the value "4" and a small up/down arrow icon. At the bottom, there are two buttons: "Confirm" and "Cancel".

Figura 3. Janela para criação de salas

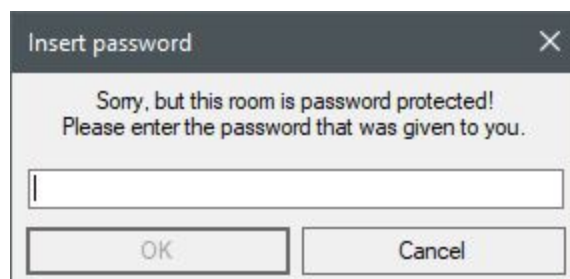
A screenshot of a dialog box titled "Insert password" with a close button (X) in the top right corner. It contains a message: "Sorry, but this room is password protected! Please enter the password that was given to you." Below the message is a single-line text input field. At the bottom, there are two buttons: "OK" and "Cancel".

Figura 4. Janela para introduzir password

Quando um cliente tenta entrar numa sala privada sem convite é aberta imediatamente uma caixa de diálogo a pedir-lhe que introduza a *password*. Se a *password* introduzida for inválida, é apresentada uma mensagem de erro. Caso contrário, é criada uma nova janela para a conversação de grupo, as mensagens anteriores são descarregadas do servidor, bem como a lista dos utilizadores que nela participam.

É possível convidar utilizadores a entrar numa sala, independentemente de ser pública ou privada (estar protegida com *password*). Os clientes convidados recebem a notificação de convite na forma de uma *MessageBox*. Caso aceitem, entram imediatamente na mesma.

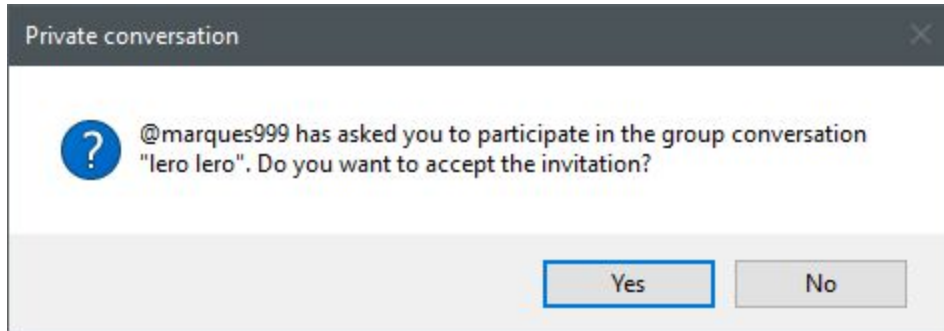


Figura 5. Janela de confirmação de convite, evidenciando que um utilizador (@marques999) convidou outro a participar numa conversação de grupo chamada “lero lero”.

O único pré-requisito no envio de convites é que os utilizadores devem estar a participar na conversa no momento para poderem convidar outros. Estes devem dirigir-se à janela principal da aplicação cliente, seleccionar um dos utilizadores da lista e carregar no botão “Invite”, sendo apresentada a seguinte janela:

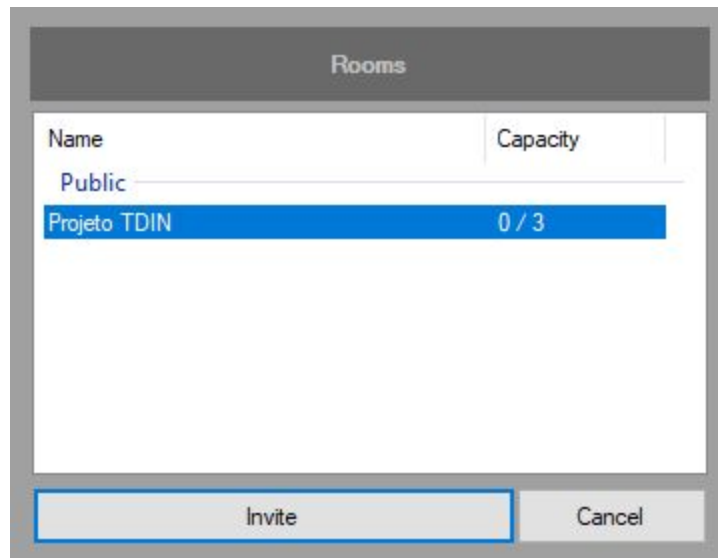


Figura 6. Janela para envio de convites aos utilizadores, evidenciando uma lista com as conversas de grupo abertas na aplicação cliente, bem como a sua lotação

A janela da conversação apresenta uma caixa de texto para envio de mensagens, um botão “Send”, uma lista com os participantes, bem como uma caixa de texto onde são registadas tanto as mensagens enviadas como as recebidas. Estas são identificadas com cores, possuindo ainda uma data de envio associada. As cores são geradas aleatoriamente para cada utilizador pela aplicação cliente sempre que esta é iniciada, a qual é comunicada aos utilizadores quando iniciam uma conversa, ou ao servidor quando entram numa sala.

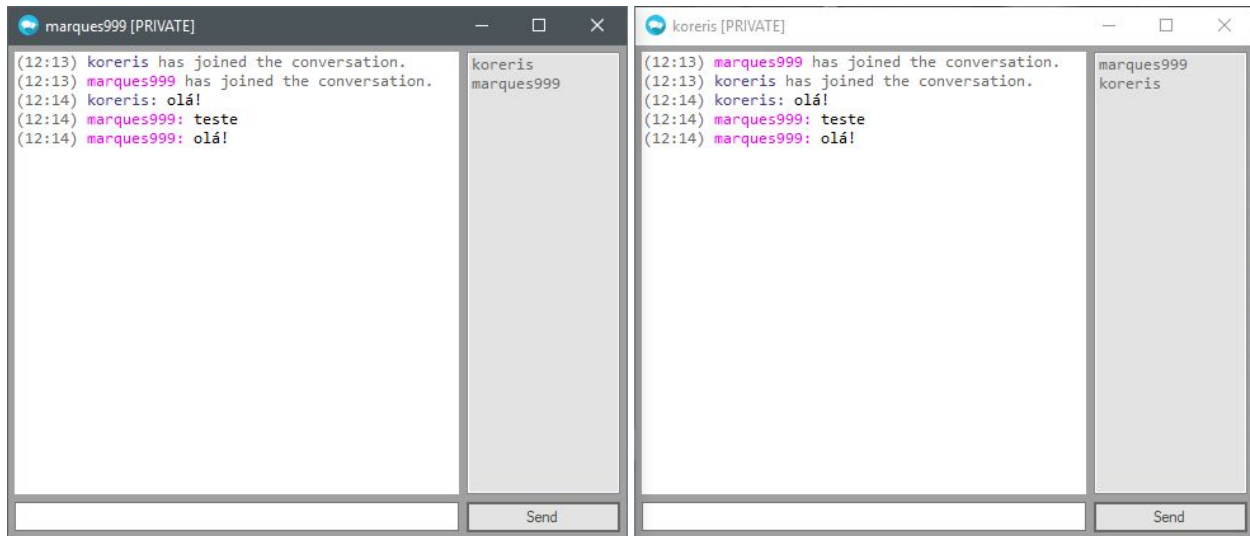


Figura 7. Janela de conversação aberta em ambos os lados da comunicação, evidenciando as mensagens, as cores dos utilizadores, as datas de envio, bem como a lista de participantes.

Qualquer utilizador poderá terminar uma conversa privada com outro utilizador fechando a janela, ação que levará ao fecho da janela correspondente no outro lado da comunicação. No caso das salas, abandonar a conversa não tem qualquer efeito para os utilizadores que nela continuem a participar. A conversa não é interrompida, sendo que estes são apenas notificados da saída do utilizador em questão. Pelo contrário, as salas de *chat* criadas pelos utilizadores persistem no servidor, isto é, não são destruídas assim que todos os participantes abandonam a mesma, nem quando fecha a aplicação do servidor. A sessão estabelecida na aplicação cliente é terminada com um *logout* na janela inicial (*lobby*), ou simplesmente fechando-a. É apresentada ao utilizador uma caixa de mensagem para este confirmar se realmente deseja terminar sessão. Caso responda afirmativamente, todas as janelas de conversa são fechadas, regressando ao ecrã de *login*, onde pode escolher sair da aplicação ou entrar com outra conta de utilizador.

2 Arquitetura

As operações de *login*, *logout*, bem como registo de um novo utilizador são realizadas entre cada cliente e um servidor único conhecido pelos clientes. Sempre que houver qualquer alteração no estado dos utilizadores ou das salas existentes no sistema, este servidor encarrega-se de notificar todos os clientes conectados de modo a atualizar as suas respetivas listas de utilizadores ativos e de salas disponíveis.

As conversações podem realizar-se diretamente entre os clientes, semelhante ao que se verifica numa rede *peer-to-peer*. Um *peer* na nossa implementação (também chamado de cliente na perspetiva do servidor central) desempenha um papel tanto de cliente como de servidor nas conversações diretas com outros clientes, na medida em que para enviar mensagens ou convites para participar numa conversa de grupo, a aplicação conecta-se ao objeto remoto disponibilizado pelo cliente com quem estabelece comunicação, ao mesmo tempo que publica também um objeto remoto ao qual outros clientes se podem conectar igualmente para enviar mensagens.

Para permitir a comunicação ponto a ponto, o servidor indica um endereço suficiente de cada cliente quando fornece ou atualiza a lista com os utilizadores ativos. Havendo vários clientes na mesma máquina, estes têm os seus objetos remotos em portas diferentes, atribuídos automaticamente aquando do arranque da aplicação. O servidor guarda as informações dos utilizadores registados, bem como das salas de *chat* criadas pelos mesmos recorrendo a uma base de dados persistente baseada na tecnologia *SQLite*.

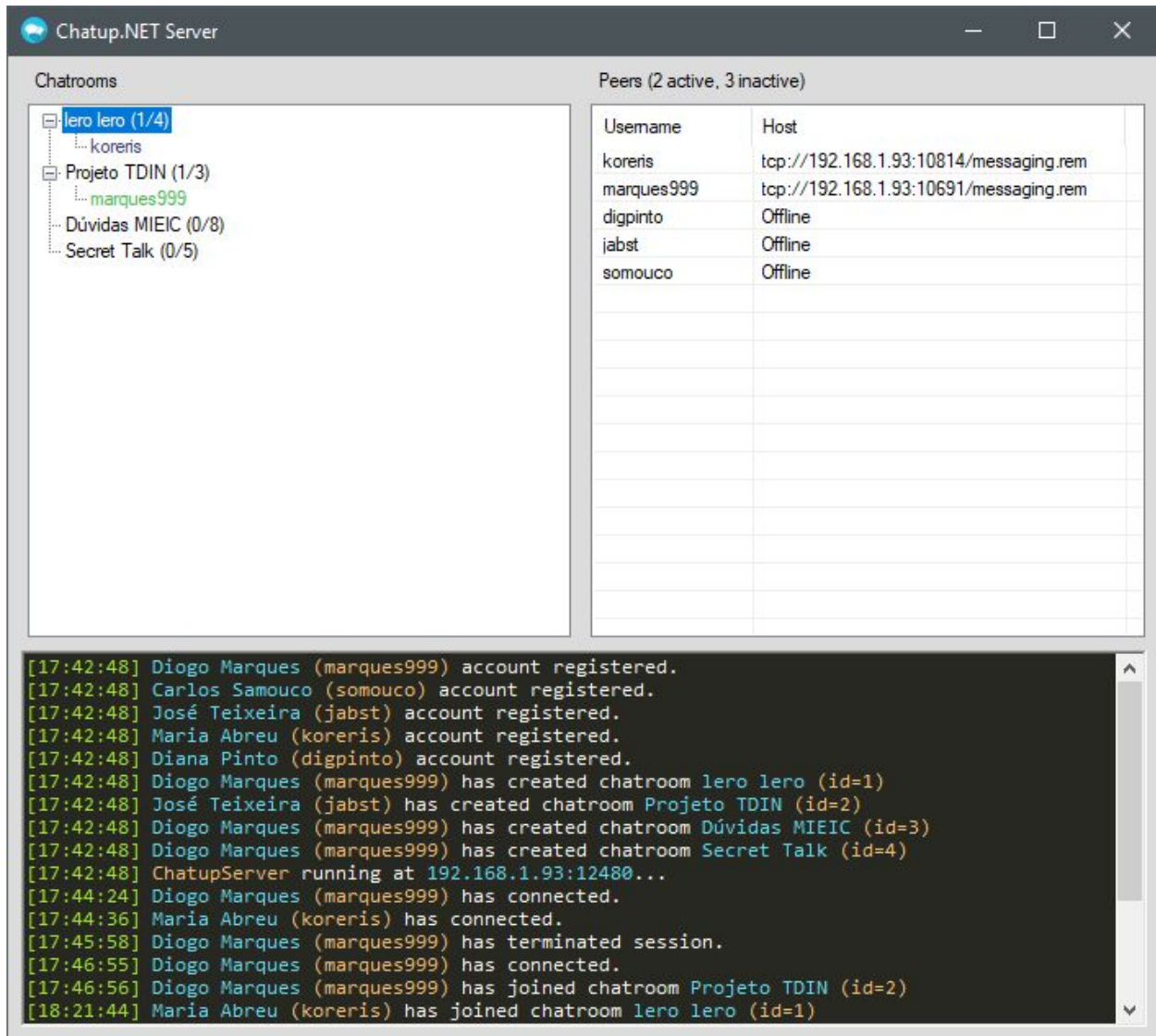


Figura 8. Janela da aplicação servidor ao ser executada, evidenciando na consola o processo de carregamento de salas e utilizadores a partir da base de dados.

É ainda possível criar salas onde vários utilizadores possam participar, bem como manter conversas com vários parceiros em simultâneo, em janelas separadas. Como as janelas são lançadas a partir da janela inicial, esta configuração garante que todas são fechadas no momento do *logout* do utilizador, ou seja, que nenhuma se encontra aberta quando este regressa à janela de autenticação. Ao contrário das conversações *peer-to-peer*, as comunicações relacionadas com os serviços de autenticação e gestão de *chatrooms* são realizadas diretamente com o servidor, implementando uma arquitetura cliente-servidor tradicional.

3 Serviços

Esta secção servirá para descrever com detalhe os diferentes serviços implementados na nossa aplicação de *chat*, as suas funcionalidades, bem como a sua contribuição individual para o correto funcionamento do sistema como um todo.

3.1 *SessionInterface*

Esta *interface* é implementada pela **aplicação servidor** e define um conjunto de métodos que podem ser invocados remotamente pelos clientes relacionados com a gestão de utilizadores, bem como eventos para que os utilizadores possam ser notificados das operações executadas pelos outros clientes, incluindo registo de novos utilizadores e autenticação dos utilizadores (operações de *login/logout*).

3.1.1 Eventos

| |
|---|
| event UserHandler OnLogin |
| <ul style="list-style-type: none">• Evento subscrito pelos clientes remotos que desejam receber notificações sempre que um utilizador se autentica no serviço;• Utilizado tanto na aplicação cliente (invocado remotamente) como na aplicação servidor (invocado localmente) para atualizar o endereço remoto dos respetivos utilizadores da lista de <i>peers</i>, passando de “Offline” para um endereço no formato hostname:porta. |
| event UserHandler OnLogout |
| <ul style="list-style-type: none">• Evento subscrito pelos clientes remotos que desejam receber notificações sempre que um utilizador termina a sua sessão no serviço;• Utilizado tanto na aplicação cliente (invocado remotamente) como na aplicação servidor (invocado localmente) para remover o endereço remoto dos respetivos utilizadores da lista de utilizadores, passando para “Offline”. |
| event UserHandler OnRegister |
| <ul style="list-style-type: none">• Evento subscrito pelos clientes remotos que desejam receber notificações sempre que um novo utilizador se regista no sistema;• Utilizado tanto na aplicação cliente (invocado remotamente) como na aplicação servidor (invocado localmente) para adicionar novos utilizadores à lista de <i>peers</i>. |

3.1.2 Métodos

| Dictionary<string, UserInformation> Users | |
|--|---|
| Descrição | Permite ao cliente obter uma lista com todos os utilizadores registados. |
| Resultado | Mapeamento dos <i>usernames</i> aos objetos contendo as informações dos utilizadores, incluindo os pares endereço IP : porta dos utilizadores ativos. |
| RemoteResponse Login(UserLogin userLogin) | |
| Descrição | Permite ao cliente iniciar sessão, realizando a sua autenticação. |
| Parâmetros | userLogin : objeto contendo as informações da sessão do utilizador. |
| Resultado | BadRequest : parâmetros em falta ou que assumem valores nulos; SessionExists : utilizador encontra-se com sessão iniciada no servidor; AuthenticationFailed : utilizador não introduziu a <i>password</i> correta; Success : utilizador autenticou-se com sucesso. |
| RemoteResponse Logout(string userName) | |
| Descrição | Permite ao cliente terminar a sua sessão na aplicação. |
| Parâmetros | userName : nome do utilizador que invocou este método. |
| Resultado | BadRequest : parâmetros em falta ou que assumem valores nulos; PermissionDenied : utilizador não possui sessão iniciada; Success : utilizador terminou sessão com sucesso. |
| RemoteResponse Register(UserForm userForm) | |
| Descrição | Permite ao cliente registar uma conta de utilizador no sistema. |
| Parâmetros | userForm : objeto contendo as informações do formulário de registo. |
| Resultado | BadRequest : parâmetros em falta ou que assumem valores null SessionExists : utilizador encontra-se atualmente com sessão iniciada no servidor AuthenticationFailed : utilizador não introduziu a password corretamente Success : utilizador autenticou-se com sucesso utilizando as suas credenciais, encontra-se agora com sessão iniciada no servidor |

3.2 LobbyInterface

Esta *interface* é implementada pela **aplicação servidor** e define um conjunto de métodos que podem ser invocados remotamente pelos clientes relacionados com a gestão de salas de *chat*, bem como eventos para que os utilizadores possam ser notificados das operações executadas pelos outros clientes, sendo estas:

- Criar salas
- Listar salas
- Apagar salas
- Consultar informações sobre determinada sala
- Obter endereço do objeto remoto de determinada sala

3.2.1 Eventos

| |
|---|
| event InsertHandler OnInsert |
| <ul style="list-style-type: none">• Evento subscrito pelos clientes remotos que desejam receber notificações sempre que um utilizador adiciona uma nova sala de <i>chat</i> ao servidor.• Utilizado tanto na aplicação cliente (invocado remotamente) como na aplicação servidor (invocado localmente) para adicionar elementos à lista de salas presente na <i>interface</i>. |
| event DeleteHandler OnDelete |
| <ul style="list-style-type: none">• Evento subscrito pelos clientes remotos que desejam receber notificações sempre que um utilizador apaga do servidor uma das salas de <i>chat</i> que criou.• Utilizado tanto na aplicação cliente (invocado remotamente) como na aplicação servidor (invocado localmente) para remover elementos da lista de salas presente na <i>interface</i>. |
| event UpdateHandler OnUpdate |
| <ul style="list-style-type: none">• Evento subscrito pelos clientes remotos que desejam receber notificações sempre que um utilizador entra ou abandona uma sala de <i>chat</i> existente no servidor.• Utilizado apenas na aplicação cliente (invocado remotamente) para atualizar os valores da lotação na lista de salas presente na <i>interface</i>. |

3.2.2 Métodos

| Tuple<RemoteResponse,Room> Insert(Room roomInformation) | |
|---|--|
| Descrição | Permite adicionar uma nova sala de <i>chat</i> ao servidor, a qual pode ser vista por todos os clientes que estejam conectados. |
| Parâmetros | roomInformation : objeto contendo todas as informações necessárias à criação de uma nova sala (nome, <i>password</i> , lotação máxima) |
| Tuple<RemoteResponse,string> Join(int roomId, string roomPassword) | |
| Descrição | Permite validar a entrada do utilizador numa sala existente no servidor, obtendo um endereço IP do objeto remoto da mesma em caso de sucesso. |
| Parâmetros | roomId : identificador único da sala de <i>chat</i> . roomPassword : palavra-passe da sala (caso esta seja privada). |
| RemoteResponse Delete(int roomId, string userName) | |
| Descrição | Método invocado remotamente pelo cliente para apagar uma sala de <i>chat</i> da qual é proprietário (deve encontrar-se necessariamente vazia). |
| Parâmetros | roomId : identificador único da sala de <i>chat</i> em questão userName : nome do utilizador que invocou este método |

3.3 RoomInterface

Esta *interface* é implementada pela **aplicação servidor** e define um conjunto de métodos que podem ser invocados remotamente pelos clientes que estejam a participar em determinada conversação de grupo ou que procurem interagir com a mesma, bem como eventos para que estes possam ser notificados das operações executadas pelos outros utilizadores nela presentes, tais como entrar numa sala de *chat*, abandonar a conversa, enviar mensagens para todos os utilizadores presentes na sala, ou até mesmo receber mensagens dos outros utilizadores.

3.3.1 Eventos

| event JoinHandler OnJoin | |
|---|--|
| <ul style="list-style-type: none">• Evento subscrito pelos clientes remotos que desejam receber notificações sempre que um utilizador entra na sala de <i>chat</i> onde estes se encontram;• Utilizado na aplicação cliente (invocado remotamente) para atualizar a <i>ListView</i> com os utilizadores existente na janela da respectiva sala;• Utilizado na aplicação servidor (invocado localmente) para atualizar a <i>TreeView</i> com as salas registadas no sistema, bem como a lotação das mesmas. | |
| event LeaveHandler OnLeave | |
| <ul style="list-style-type: none">• Evento subscrito pelos clientes remotos que desejam receber notificações sempre que um utilizador abandona a sala de <i>chat</i> onde este se encontram;• Utilizado na aplicação cliente (invocado remotamente) para atualizar a <i>ListView</i> com os utilizadores existente na janela da respectiva sala;• Utilizado na aplicação servidor (invocado localmente) para atualizar a <i>TreeView</i> com as salas registadas no sistema, bem como a lotação das mesmas. | |
| event MessageHandler OnSend | |
| <ul style="list-style-type: none">• Evento subscrito pelos clientes que desejam receber notificações sempre que um utilizador envia uma mensagem de texto para uma das salas onde estes se encontram;• Utilizado na aplicação cliente (invocado remotamente) para atualizar a <i>RichTextBox</i> com novas mensagens (se for recebido pelos outros utilizadores) como confirmação que a mensagem foi recebida pelos demais utilizadores presentes na sala (se for recebido pelo autor da mensagem). | |

3.3.2 Métodos

| Tuple<RemoteResponse,MessageQueue> Join(UserProfile userProfile) | |
|--|---|
| Descrição | Permite ao cliente entrar numa sala existente no servidor. No caso de uma sala privada, assume-se que este utilizador tenha validado previamente a respetiva <i>password</i> junto do servidor. |
| Parâmetros | userProfile : objeto contendo informações relativas ao utilizador que pretende entrar na sala (<i>username</i> , cor das mensagens) |

| RemoteResponse Send(RemoteMessage remoteMessage) | |
|---|---|
| Descrição | Permite ao cliente presente numa sala enviar uma mensagem. |
| Parâmetros | remoteMessage : objeto contendo a mensagem enviada pelo utilizador (bem como informações associadas, tais como autor, <i>timestamp</i>) |
| RemoteResponse Leave(string userName) | |
| Descrição | Permite ao cliente presente numa sala abandonar a mesma. |
| Parâmetros | userName : nome do utilizador que invocou este método |

3.4 *MessageInterface*

Este serviço é implementado na **aplicação cliente** e define um conjunto de métodos que podem ser invocados remotamente pelos clientes relacionados com o estabelecimento de conversações *peer-to-peer* com outros clientes, bem como eventos para que ambos possam ser notificados das operações executadas.

3.4.1 Eventos

| event InviteHandler OnInvite |
|---|
| Evento utilizado pela aplicação cliente (invocado localmente) para receber notificações na <i>interface</i> sempre que recebe um convite dos outros utilizadores para participar numa conversa; |
| event MessageHandler OnReceive |
| Evento utilizado pela aplicação cliente (invocado localmente) para receber na respectiva janela as mensagens enviadas pelos utilizadores em conversas privadas com este |
| event ConnectHandler OnConnect |
| Evento utilizado pela aplicação cliente (invocado localmente) para receber notificações na <i>interface</i> sempre que outro utilizador tenta iniciar uma conversa privada com esta, levando ao lançamento de uma janela de conversação |

event DisconnectHandler OnDisconnect

Evento utilizado pela aplicação cliente (invocamento localmente) para receber notificações na *interface* sempre que um utilizador fecha a janela da conversa remotamente, levando também ao fecho da janela localmente.

3.4.2 Métodos

| RemoteResponse Disconnect(string userName) | |
|---|--|
| Descrição | Permite ao cliente abandonar uma conversa privada. |
| Parâmetros | userName : nome do utilizador que fechou a janela da conversa. |
| RemoteResponse Send(RemoteMessage remoteMessage) | |
| Descrição | Permite ao cliente enviar mensagens aos utilizadores com os quais tem conversas privadas abertas. |
| Parâmetros | remoteMessage : objeto contendo a mensagem enviada pelo utilizador (bem como informações associadas, tais como autor, <i>timestamp</i>). |
| RemoteResponse Invite(RoomInvitation roomInvitation) | |
| Descrição | Permite ao cliente convidar outros utilizadores a participar numa conversa de grupo da qual faz parte. |
| Parâmetros | roomInvitation : objeto contendo todas as informações relativas ao convite enviado pelo utilizador (<i>username</i> da pessoa que convidou, identificador da sala para onde foi convidado, <i>password</i> da sala). |
| Tuple<RemoteResponse,UserProfile> Connect(UserProfile userProfile, string userHost) | |
| Descrição | Permite ao cliente iniciar uma conversa privada com outro utilizador. |
| Parâmetros | userProfile : objeto contendo informações relativas ao utilizador que pretende iniciar uma conversa privada (<i>username</i> , cor das mensagens); userHost : endereço remoto do utilizador ao qual pretendemos ligar. |

4 Diagramas UML

A aplicação encontra-se estruturada em três componentes diferentes, cada uma representada por um projeto diferente no *Visual Studio* que no conjunto constituem a *solution* da aplicação, como pode ser observado no seguinte diagrama:

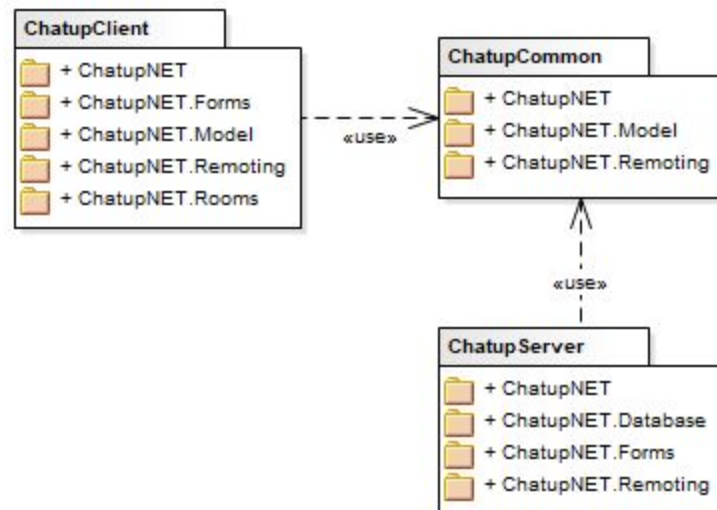


Diagrama da solution ChatupNET, evidenciando as relações de dependência entre os vários projetos, bem como os namespaces existentes em cada um.

4.1 ChatupClient

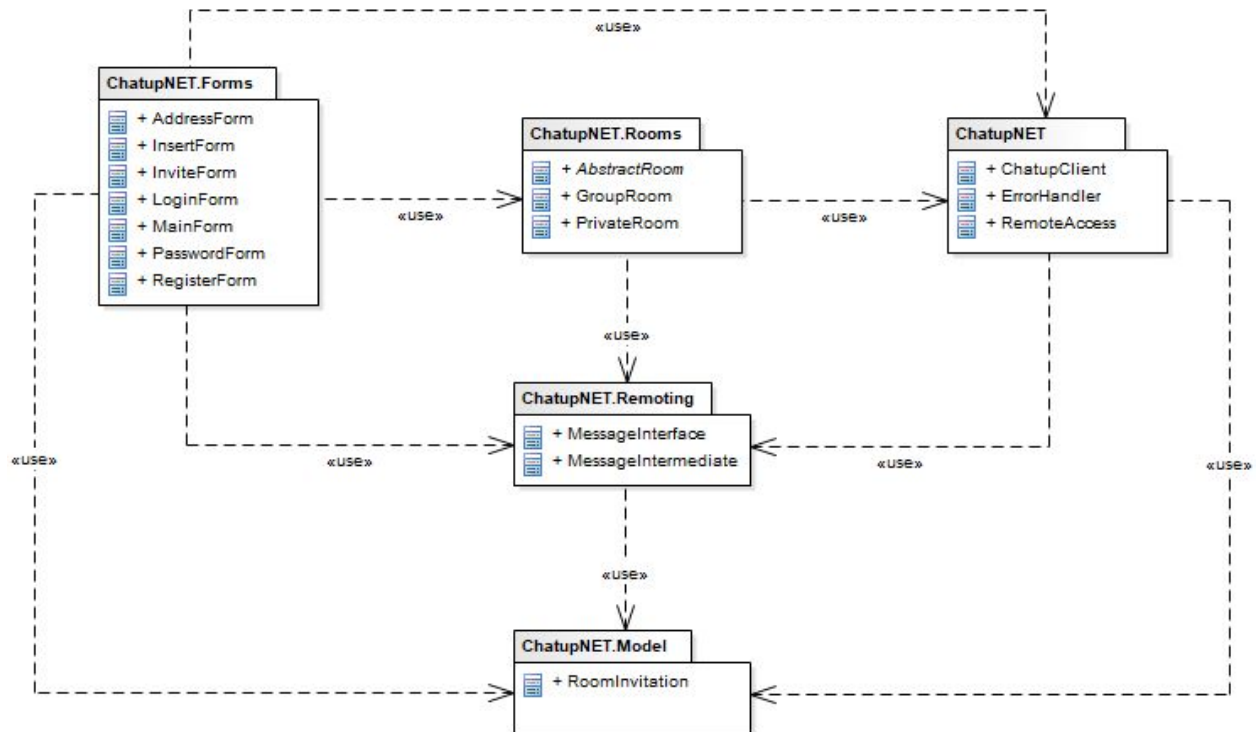


Diagrama de classes do projeto ChatupClient, evidenciando as relações de dependência entre os vários namespaces, bem como a sua organização em classes.

| Namespace | Descrição |
|---------------------------|---|
| <i>ChatupNET</i> | Funcionalidades necessárias ao funcionamento da aplicação cliente; contém ainda uma classe <i>singleton</i> que guarda informações relativas ao estado da mesma. |
| <i>ChatupNET.Forms</i> | <i>Forms</i> utilizados na <i>interface</i> gráfica da aplicação, tais como a janela de <i>login</i> , a janela inicial da aplicação, a janela de registo um novo utilizador, a janela para introdução da <i>password</i> da sala, etc... |
| <i>ChatupNET.Model</i> | Estruturas de informação utilizadas nas chamadas <i>remoting</i> para troca de mensagens entre os clientes, bem como enviar convites. |
| <i>ChatupNET.Remoting</i> | Disponibiliza <i>serviços</i> de <i>remoting</i> utilizados pelos clientes para iniciar conversas privadas e trocar mensagens entre eles. |
| <i>ChatupNET.Rooms</i> | <i>Forms</i> utilizados na <i>interface</i> gráfica da aplicação, tais como a janela de conversação <i>peer-to-peer</i> ou a janela de conversação em grupo. |

4.2 ChatupCommon

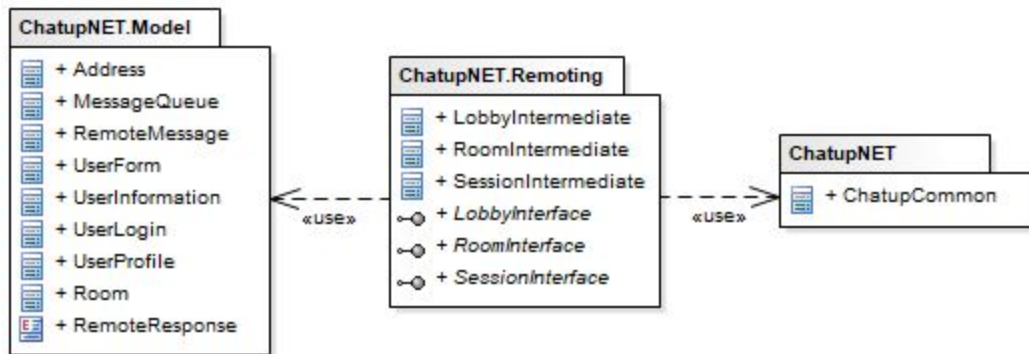


Diagrama de classes do projeto ChatupCommon, evidenciando as relações de dependência entre os vários namespaces, bem como a sua organização em classes.

| Namespace | Descrição |
|---------------------------|---|
| <i>ChatupNET</i> | Implementa funcionalidades comuns tanto à aplicação cliente como à aplicação servidor. |
| <i>ChatupNET.Model</i> | Define várias estruturas de informação utilizadas na comunicação cliente-servidor através de chamadas remotas, tais como formulários de registo, formulário de <i>login</i> , mensagens de texto, etc... |
| <i>ChatupNET.Remoting</i> | Define um conjunto de <i>interfaces</i> e classes <i>proxy</i> utilizadas pelos clientes para aceder aos serviços de <i>remoting</i> disponibilizados pelo servidor, pelo servidor na implementação desses mesmos serviços. |

4.3 ChatupServer

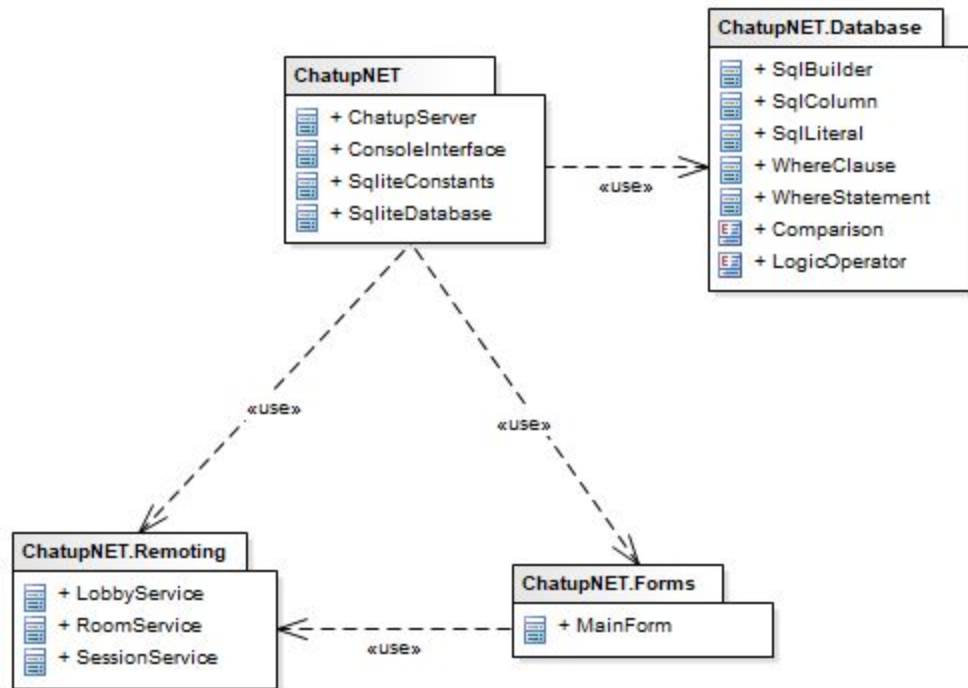


Diagrama de classes do projeto ChatupServer, evidenciando as relações de dependência entre os vários namespaces, bem como a sua organização em classes.

| Namespace | Descrição |
|---------------------------|---|
| <i>ChatupNET</i> | Funcionalidades necessárias ao funcionamento do servidor; contém uma classe <i>singleton</i> que guarda informações relativas ao estado da mesma; implementa ainda várias operações que permitem aceder e executar <i>queries</i> sobre a base de dados SQL da aplicação. |
| <i>ChatupNET.Database</i> | Define uma <i>interface</i> intuitiva para construção de <i>queries</i> em SQL. |
| <i>ChatupNET.Forms</i> | <i>Forms</i> utilizados na <i>interface</i> gráfica da aplicação servidor, tais como a janela inicial, que permite a administração do mesmo. |
| <i>ChatupNET.Remoting</i> | Define um conjunto de <i>interfaces</i> e classes <i>proxy</i> utilizadas pelos clientes para aceder aos serviços de <i>remoting</i> disponibilizados pelo servidor, pelo servidor na implementação desses mesmos serviços. |

5 Conclusão

No final do desenvolvimento deste projeto foi possível adquirir competências básicas no domínio de .NET Remoting e da linguagem de programação C#, como também aprofundar a experiência na conceção de aplicações com interfaces gráficas e em aplicações desenhadas com uma arquitetura cliente-servidor.

Como resultado final, a aplicação desenvolvida é capaz de permitir a vários utilizadores conectarem-se a um servidor centralizado e a partir daí comunicarem com outros utilizadores recorrendo a um sistema de mensagens privadas *peer-to-peer* ou a salas de *chat* que possuem capacidade para dois ou mais utilizadores realizarem trocas de mensagens em simultâneo. Estas salas de *chat* podem ser privadas ou públicas, sendo a única coisa que as distingue a restrição do seu acesso por uma palavra-passe. Os utilizadores conseguem registar-se no servidor e após o seu *login* o servidor regista informação relativa aos mesmos para que seja possível posteriormente dois utilizadores estabelecerem uma ligação bilateral para troca de mensagens sem que seja necessário um mecanismo de *middle-man*.

Como possível acréscimo às funcionalidades da aplicação poder-se-ia ter implementado um mecanismo de envio de ficheiros de imagem e ficheiros binários para enriquecer a experiência de comunicação. Outra vertente, não ligada diretamente à *user experience* nem às funcionalidades da aplicação, seria portar a aplicação para que seja possível correr sobre a Internet e não estar limitada a uma rede local, o que leva a um segundo ponto que seria de adicionar na GUI da aplicação do servidor para dar a escolher ao utilizador em qual *interface* de rede desejaria lançar o servidor.

6 Recursos

Segue-se uma lista de todas as referências bibliográficas consultadas, bem como do *software* utilizado ao longo do desenvolvimento deste trabalho:

6.1 Referências bibliográficas

- ❖ Apontamentos das aulas teórico-práticas sobre arquitecturas de sistemas distribuídos, alguns aspectos da linguagem C# e tecnologia *.NET Remoting* disponibilizados no sítio oficial da unidade curricular (<https://paginas.fe.up.pt/~apm/TDIN/main.html>)
- ❖ **“All you need to know about .NET Remoting - CodeProject”**
(<http://www.codeproject.com/Articles/29945/All-you-need-to-know-about-NET-Remoting>)
- ❖ **“.NET Remoting with an easy example - CodeProject”**
(<http://www.codeproject.com/Articles/14791/NET-Remoting-with-an-easy-example>)
- ❖ **“Microsoft Developer Network - Remoting Example: Dynamic Publication”**
([https://msdn.microsoft.com/en-us/library/1t63e8ff\(v=vs.100\).aspx?cs-lang=csharp](https://msdn.microsoft.com/en-us/library/1t63e8ff(v=vs.100).aspx?cs-lang=csharp))
- ❖ **“SelectQueryBuilder: Building complex and flexible SQL queries/commands from C# - CodeProject”**
(<http://www.codeproject.com/Articles/13419/SelectQueryBuilder-Building-complex-and-flexible-S>)
- ❖ **“System.Data.SQLite: Home”**
(<https://system.data.sqlite.org/index.html/doc/trunk/www/index.wiki>)

6.2 Software utilizado

- ❖ **“DB Browser for SQLite”** (<http://sqlitebrowser.org>)
DB Browser for SQLite is a high quality, visual, open source tool to create, design, and edit database files compatible with SQLite, targeting users and developers wanting to create databases, search, and edit data.

❖ **“System.Data.SQLite”** (<https://system.data.sqlite.org/index.html/doc/trunk/www/index.wiki>)

System.Data.SQLite is an SQLite provider for .NET Runtime that implements every feature of the underlying SQLite database management system without omission. Written from scratch specifically for ADO.NET, it implements all the base classes and features introduced in this framework, including automatic transaction enlistment.

❖ **“Visual Studio Enterprise 2015”** (<https://www.visualstudio.com/vs>)

Microsoft Visual Studio is an integrated development environment (IDE) from Microsoft, used to develop computer programs for Windows, as well as websites, web applications, web services and mobile applications. Visual Studio integrates Microsoft software development platforms such as Windows API, Windows Forms, Windows Communication Foundation (WCF), Windows Presentation Foundation (WPF), Windows Store and Microsoft Silverlight. It can produce both native code and managed code.