

Semestrální práce KIV-UPS

# 7. Chatovací systém

Klient-Server aplikace

**Jméno:** Pavel Lorenz

**Email:** pavel.lorenz@students.zcu.cz

**Univerzitní číslo:** A10B0061K

## Obsah

7. Chatovací systém.....	2
Zadání .....	2
Programátorská dokumentace.....	2
Technologie .....	2
Knihovna (ChatLibrary).....	2
ChatKlient .....	5
ChatServer .....	5
Uživatelská dokumentace .....	7
ChatKlient .....	9
ChatServer .....	8
Odkazy na použité zdroje .....	<b>Chyba! Záložka není definována.</b>
Závěr .....	<b>Chyba! Záložka není definována.</b>

## 7. Chatovací systém

### Zadání

Chatovací systém. Realizujte programy serveru a klienta pro chatování. Chatovací server a klient bude podporovat přihlášení uživatele pod přezdívkou, komunikaci s ostatními uživateli, pouze s jedním definovaným uživatelem a odhlášení uživatele.

Protokol bude obsahovat příkazy LOGIN, LOGOUT, ALL\_MSG, PRIV\_MSG, USERS, PING a odpovědi OK a ERR.

### Programátorská dokumentace

#### Technologie

Chatovací systémy lze realizovat pravděpodobně v jakémkoliv vyspělejším jazyku (např. Ansi C, C++, Java, C#, aj.). Protože se v práci nejvíce soustředím na technologii .net, tak jsem si po dohodě, zvolil programovací jazyk C#.

Mým cílem bylo, vytvořit jednoduše ovládatelnou základní knihovnu pro práci se sockety a jako nadstavbu nad touto knihovnou postavit klienta a server.

Server jsem realizoval jako konzolovou aplikaci, která je velice jednoduchá a hlavní podíl práce bude v knihovně.

Klienta jsem realizoval pomocí technologie WPF jako okenní aplikaci, aby práce s ním byla více uživatelsky přívětivá. Důraz jsem kladl na přehlednost a robustnost.

#### Knihovna (ChatLibrary)

##### Požadavky

- Zapouzdřenost
- Jednoduchost
- Vstupy ovládané pomocí metod
- Výstupy realizované pomocí událostí
- Základní komunikace, pouze odesílání a přijímání textových zpráv

##### Realizace

Knihovna „ChatLibrary“ je oddělena od klienta i serveru. Je to samotná dll knihovna. Pro přehlednost jsem ji rozdělil pomocí namespaces je do jednotlivých bloků.

- **Arguments** (argumenty událostí, které mohou vyvstat)
- **Converters** (pomocné třídy pro převod text. řetězce na pole bytů a zpět)
- **Messages** (objekty předávané chatem)
- **ReadingLoop** (smyčka, která vyčítá zprávy na socketu)
- **UserManagements** (třídy reprezentující uživatele chatu)

V samotném kořenovém namespace se nacházejí klíčové třídy:

- **ChatSocket** (abstraktní třída implementující společné prvky serveru i klienta)
- **SimpleChatClient** (jednoduchá implementace klienta)
- **SimpleChatServer** (jednoduchá implementace serveru)

Vše jsem se snažil programovat přes rozhraní, takže není nutné používat mojí implementaci. Kdokoliv si může vytvořit vlastní implementaci, např. vyčítací smyčky a použít ji.

Pro síťovou komunikaci bude ostatním programátorům plně postačovat připojit reference na ChatLibrary a vytvořit instance SimpleChatClienta/SimpleChatServeru.

### *SimpleChatClient*

Tato třída implementuje přijímání a odesílání textových zpráv.

#### Konstruktory

```
SimpleChatClient(IPAddress ip, int port)
```

Základní konstruktor třídy. Na předané ip adrese a portu bude klient očekávat server.

```
SimpleChatClient(IPAddress ip, int port, IByteToStringConverter converter,
IReadingLoopCreator creator)
```

Nejrozšířenější konstruktor, který vyžaduje instanci konvertru a tovární třídy na „vyčítací smyčky“.

#### Metody

```
virtual void Connect()
```

Metoda, která je v základu virtuální, abychom ji v případných potomcích mohli překrýt, a jinak se postará o spojení klienta se serverem.

```
virtual void SendMessage(string message)
```

Metoda je opět virtuální a stará se o posílání textových zpráv na server.

```
virtual void Close()
```

Poděděná metoda, která ukončuje klienta.

#### Události

```
event SocketHandler ServerSocketConnected
```

Událost, která je odpálena po připojení na server. V argumentu má socket klienta.

```
event SimpleMessageHandler MessageReceived
```

Poděděná událost, která je odpálena při přijetí zprávy. Zpráva je předána v argumentu.

```
event SocketExceptionHandler ReadingInterrupted
```

Poděděná událost, která je odpálena při chybě objevené při čtení zprávy. Argument obsahuje výjimku a socket, ze kterého bylo čteno.

## SimpleChatServer

Tato třída implementuje přijímání klientů a odesílání zpráv na jednotlivé klienty.

### Konstruktory

```
SimpleChatServer(IPAddress ip, int port)
```

Základní konstruktor třídy. Na předané ip adrese a portu bude server očekávat příchozí spojení.

```
SimpleChatServer(IPAddress ip, int port, IByteToStringConverter converter,
IReadingLoopCreator creator)
```

Nejrozšířenější konstruktor, který vyžaduje instanci konverteru a tovární třídy na „vyčítací smyčky“.

### Metody

```
virtual void StartListen()
```

Virtuální metoda, která započne naslouchání na přednastaveném portu.

```
virtual void SendMessage(Socket socket, string message)
```

Virtuální metoda, která odesílá zprávy na zadaný socket.

```
virtual void Close()
```

Poděděná metoda, která ukončuje klienta.

### Události

```
event SocketHandler SocketAccepted
```

Událost, která je odpálena po připojení na klienta. V argumentu má socket klienta.

```
event SimpleMessageHandler MessageReceived
```

Poděděná událost, která je odpálena při přijetí zprávy. Zpráva je předána v argumentu.

```
event SocketExceptionHandler ReadingInterrupted
```

Poděděná událost, která je odpálena při chybě objevené při čtení zprávy. Argument obsahuje výjimku a socket, ze kterého bylo čteno.

## Shrnutí

Pomocí výše uvedených metod a událostí jsme snadno schopni realizovat jakoukoliv komunikaci po síti. Stačí si připravit pevně daný, známý formát zprávy, zapsat ho do objektu, serializovat a odeslat jako textovou zprávu.

Na druhé straně, díky známému formátu budeme schopni objekt deserializovat a data přečíst.

Klient je realizován pomocí dvou vláken. Hlavní vlákno si drží například konzole a pomocí SendMessage může zprávy posílat. Vedlejší vlákno neustále vyčkává na socketu a čte příchozí zprávy.

Server je vícevláknový. V hlavním vlákne neustále čeká na příchozí spojení a jakmile se spoj akceptován, tak vytvoří čtecí vlákno a opět vyčkává na příchozí zprávy od klienta. Pro každého klienta je vlákno nové.

## ChatServer

Server je postavený na konzolové aplikaci. Pro samotnou komunikaci používá knihovnu ChatLibrary. Prakticky jen vyhodnocuje příchozí zprávy a následně je přeposílá na místo určení.

Hlavní vlákno pouze vyčkává na příchozí spojení, a pokud nastane, tak vytvoří instanci nového vlákna, které v nekonečné smyčce čeká na příchozí zprávy.

Pokud je příchozí komunikace akceptována, tak ještě pořád nelze komunikovat se serverem a ten odpovídá na příchozí zprávy z nepřihlášeného socketu buď hlášením „*Unknown format of message*“ v případě, kdy poslaný objekt není typu ChatMessage. Při splnění daného formátu potom „*You must login before*“, kde v podstatě nutí uživatele, aby se přihlásil.

Server rozlišuje několik typů zpráv.

### LOGIN

Objekt nesoucí zprávu v těle obsahuje jméno, pod kterým bude uživatel přihlášen. Server vytvoří instanci LoggedUser, kterou přidá do kolekce registrovaných uživatelů a údaj o přihlášení se odešle všem registrovaným uživatelům.

### LOGOUT

Tento příkaz je implementován velmi podobně jako „login“. Uživatel odešle „logout“ na server. Server si zjistí, o koho šlo, vyřadí ho ze seznamu přihlášených a odešle tuto zprávu s údajem o odhlášeném uživateli na všechny zbývající.

### ALL\_MSG

Příchozí zpráva tohoto typu je serverem přeposílána na všechny registrované uživatele.

### PRIV\_MSG

Příchozí zpráva je odeslána pouze odesilatelci a cílovému uživateli.

### USERS

Server odesilatelci vrátí seznam všech registrovaných uživatelů. Tato kolekce je serializovaná a odeslána v těle zprávy.

### PING

Server kontroluje, jestli je cílový uživatel nulový nebo ne. Pokud ano, tak byl ping cílen na server. Server ve zprávě otočí odesilatele s cílem a zprávu odešle.

### Výjimky

Ve smyčce, která neustále vyčítá zprávy ze socketu, se mohou objevit chyby, např. při pádu klienta a tyto výjimky jsou odchyťovány a odpalovány pomocí jako události „ReadingInterrupted“.

Server je na této události registrován a všechny výjimky vypíše do konzole, uživatele odpojí ze seznamu a rozešle tuto zprávu všem klientům.

## ChatKlient

Klient je reprezentován pomocí „tlustého“ klienta, tedy desktopovou aplikací. Pro psaní klienta jsem měl na výběr mezi WinForm a WPF technologií.

Protože se velice zaměřuji na WPF problematiku a znám ji více než WinForms, tak jsem si vybral WPF. Je to prakticky nástupce WinForm technologie. Veškerá grafika je vektorová a hardwarově akcelerovaná.

Aby byl klient snadno modifikovatelný, využil jsem návrhového vzoru M-V-VM (Model-View-ViewModel), což je specifický vzor vycházející z tradičního návrhu MVC (Model-View-Controller) a je „na míru“ ušit přímo pro WPF technologii.

Hlavní výhoda tohoto návrhového vzoru je oddělení prezenční vrstvy od dat. View jako takové pouze data zobrazuje (binding) a veškeré akce jsou prezentovány příkazy (commands).

Lze tak snadno „zahodit“ View vrstvu a nahradit ji jinou, stačí pouze dodržet jmennou konvenci vlastností, které jsou zobrazovány a příkazů, které využíváme.

Aplikace obsahuje pouze dvě okna. Hlavní okno umožňuje kompletní ovládání chatu. Okénko „Login“ pak usnadňuje konfiguraci serveru, na který se hlásíme.

Všechny tlačítka jsou řízeny tzn. Guard vlastnostmi, které vždy kontrolují, jestli lze daná akce provést. Pokud nelze, bude tlačítko disabled.

Např. pokud nebudeme přihlášení, tak nemůžeme odesílat zprávy, protože instance pro komunikaci se serverem je prázdná. Tlačítko „Odeslat“ je neaktivní (šedivé).

Veškeré aplikační logika klienta je zapsaná v ShellViewModel třídě. Jednotlivé příkazy jsou implementovány pomocí akcí a ty se starají o komunikaci se serverem.

Stejně jako u serveru je i zde „vyčítací“ smyčka, která neustále čeká na socketu a přijímá zprávy. Ty odpalují událost „MessageReceived“. I klient je na této události registrován a všechny příchozí zprávy jsou zpracovávány v metodě „OnMessageReceived“.

## LOGIN

Příchozí zpráva „login“ je zkontrolována, jestli jde o přihlášení nového uživatele nebo sebe samotného. Pokud je to nový uživatel, tak je tělo zprávy deserializováno a instance objektu „LoggedUser“ je přidána do kolekce přihlášených uživatelů.

Pokud jde o naše prvotní přihlášení, tak deserializovaná instance je uložena do vlastnosti CurrentUser odkud si klient čte např. nickname a server je požádán a kompletní seznam uživatelů příkazem USERS.

## LOGOUT

Deserializovaný objekt „LoggedUser“ je odstraněn z kolekce přihlášených uživatelů.

## ALL\_MSG

Zpráva je uložena do kolekce přijatých zpráv.

### ***PRIV\_MSG***

Analogicky jako zprávy od všech. Zpráva je uložena do kolekce přijatých zpráv.

### ***USERS***

Tělo přijaté zprávy je deserializováno, čímž získáme kolekci přihlášených uživatelů. Touto kolekcí přepíšeme naši stávající kolekci.

Následně ještě přiložíme prázdného uživatele, který reprezentuje zprávu pro všechny.

### ***PING***

Zpracování této zprávy je komplikovanější. Nejprve zjišťujeme, jestli šlo o zpráv vyslanou od nás. Pokud ne, a pouze někdo kontroluje odezvu vůči nám, tak se ve zprávě vymění odesílatel s cílovým uživatelem a zpráva se opět odešle.

Pokud jde o zprávu od nás, tak kontrolujeme, na koho byla zpráva cílená. Uživateli je umožněno kontrolovat odezvu pouze vůči ostatním uživatelům, odezvu vůči serveru si kontroluje klient sám.

Pokud jde o uživatelskou zprávu, je uživatel informován o výsledku pomocí přidané zprávy do kolekce „Message“. Pokud jde o kontrolu odezvy vůči serveru, tak je tato zpráva uložena do vlastnosti CheckMessage a zobrazena ve status panelu.

Aplikace si sama kontroluje odezvu vůči serveru každých 20 sec.

## **Uživatelská dokumentace**

### ***Build aplikace***

Pro build aplikace je nutné mít nainstalované Visual Studio 2010 nebo 2012.

- 1) Otevřeme solution **Palo-Chat-v10.sln** (nebo v12 pro VS 2012)
- 2) Přepneme build mód na „Release“
- 3) Stiskneme „F6“ nebo v menu „Build->Build Solution“
- 4) V kořenovém adresáři projektu nalezneme složku „Bin“ a v ní složku „Release“
  - a. „ComplexChatClientExample\Bin\Release“ pro klienta
  - b. „ComplexChatServerExample\Bin\Release“ pro server

Zde jsou naše zkompilevané programy.

### ***Požadavky na spuštění***

Ke spuštění aplikace je nutné mít:

- Windows XP a vyšší
- .net Framework 3.5 SP1



## ChatServer

Chat zapneme pomocí exe souboru „**ComplexWpfChatServerExample.exe**“.

Jeho konfigurace je možná dvěma způsoby.

- 1) Pomocí příkazové řádky
- 2) Pomocí konfiguračního souboru

### *Příkazová řádka*

První argument je název serveru, druhý je port.

Příklad:

**ComplexWpfChatServerExample.exe localhost 5000**

Server se pokusí naslouchat na adrese „localhost“ a portu „5000“.

### *Konfigurační soubor*

Chat lze konfigurovat pomocí souboru „**ComplexWpfChatServerExample.exe.config**“ Konfigurační soubor vypadá takto.

```
<?xml version="1.0"?>

<configuration>

  <appSettings>

    <add key="ServerAddress" value="localhost"/>

    <add key="ServerPort" value="5000"/>

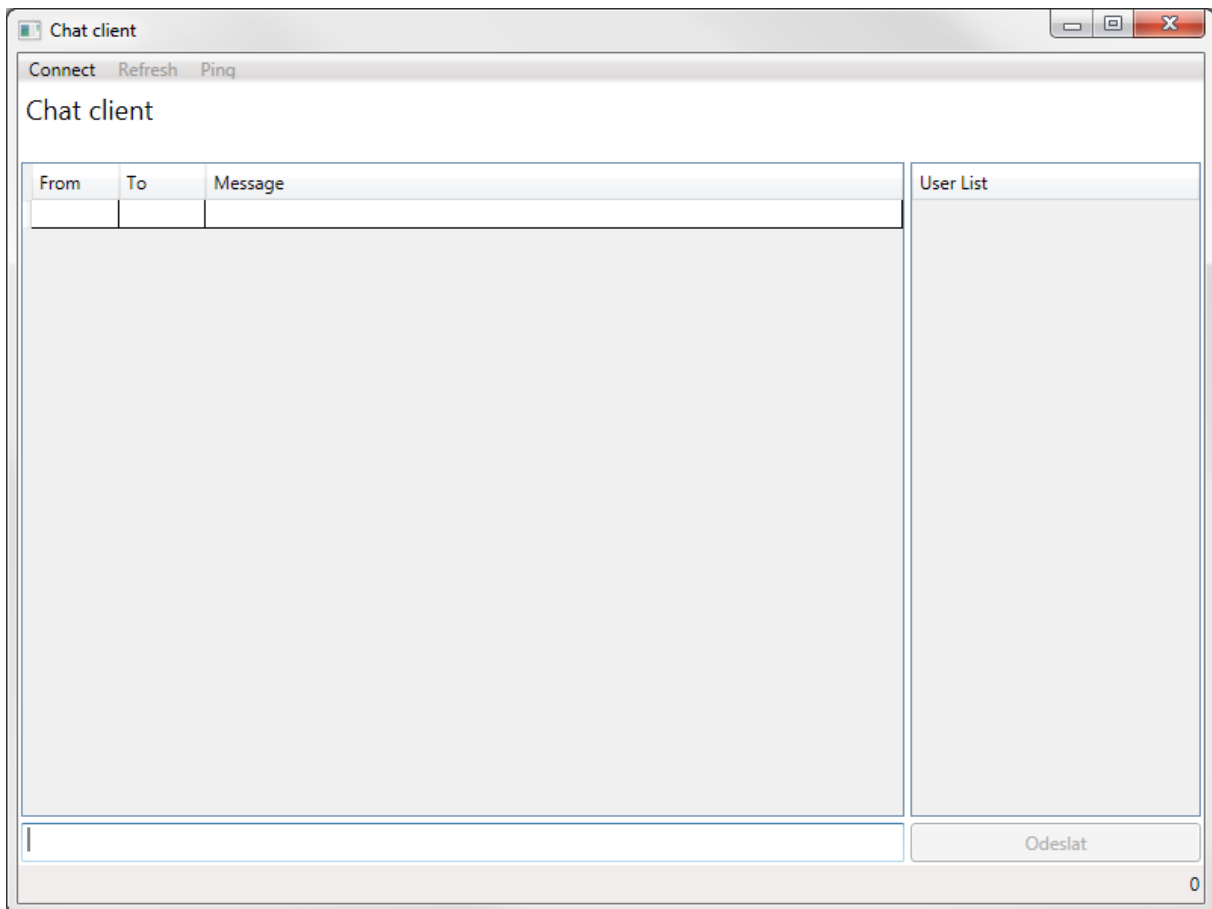
  </appSettings>

<startup><supportedRuntime version="v2.0.50727"/></startup></configuration>
```

Zde můžeme upravit klíče „**ServerAddress**“ a „**ServerPort**“ a aplikaci následně spustit pomocí poklepání na exe soubor.

## ChatKlient

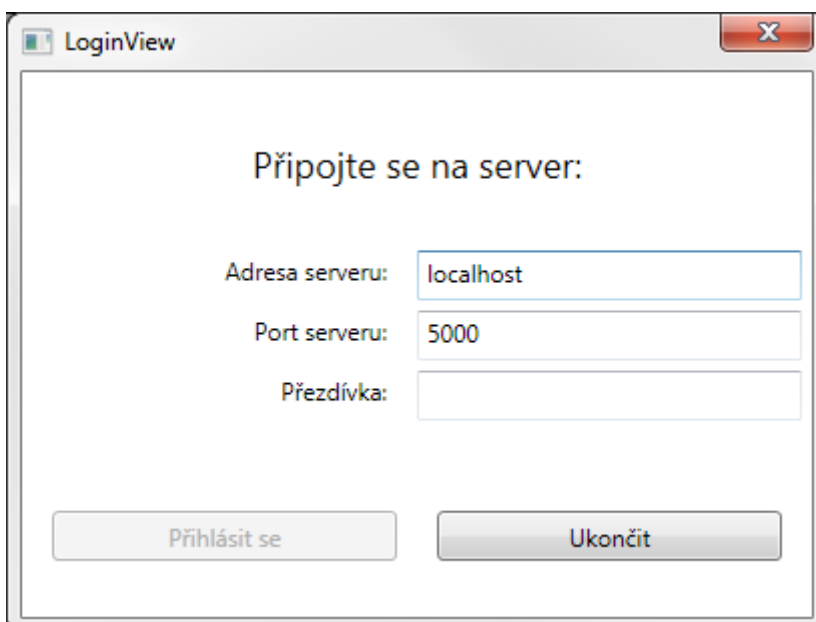
Klientskou aplikaci spustíme poklepnutím na soubor „ComplexWpfChatClientExample.exe“.



Aplikace je velice robustní, takže nedovolí uživateli něco, čímž by zapříčinil pád klienta (ne-li serveru).

## Přihlášení

V aplikaci klikneme na tlačítko „Connect“.



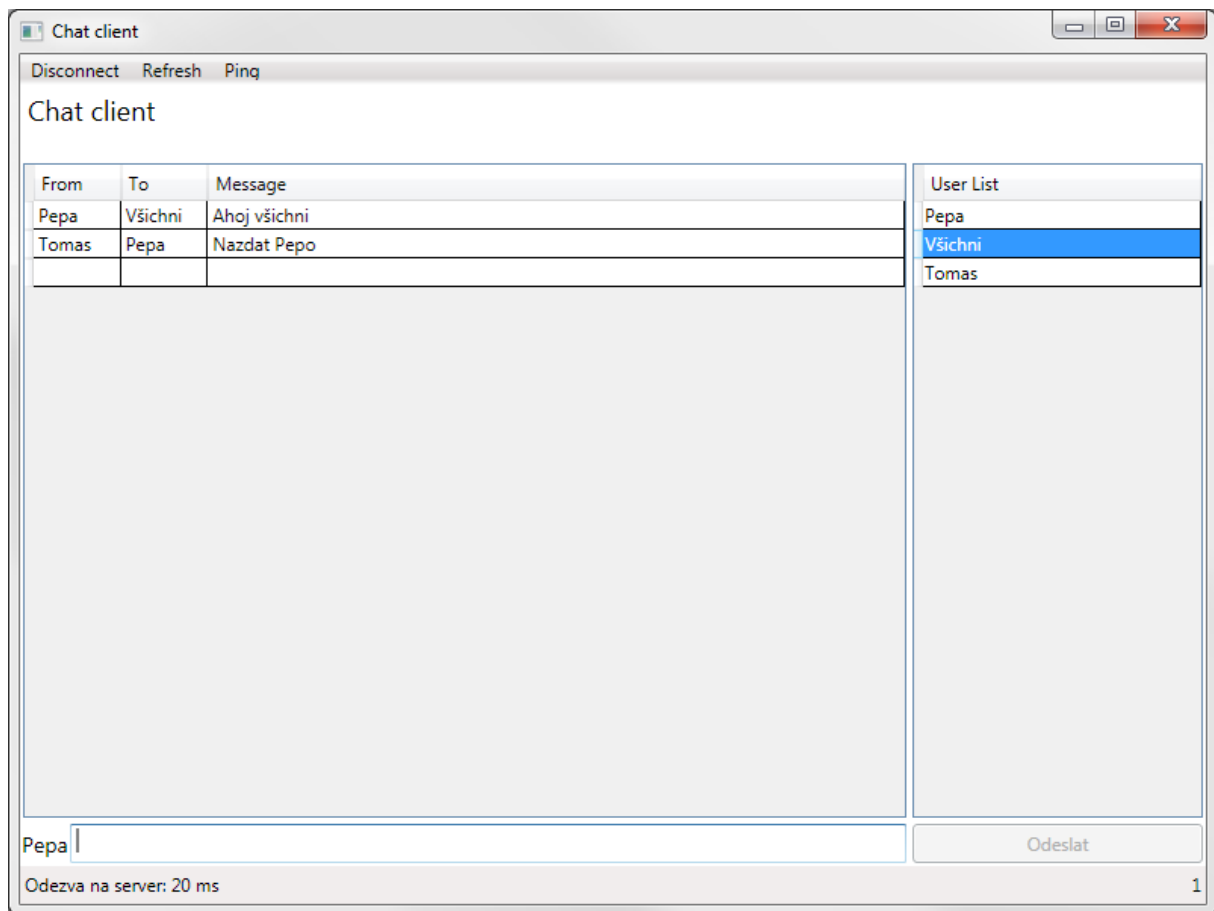
- Do políčka „Adresa serveru“ napíšeme adresu serveru.
- Do políčka „Port serveru“ napíšešeme port serveru.
- Do políčka „Přezdívka“ napíšeš svůj přezdívku.

Po splnění výše uvedených požadavků se nám teprve aktivuje tlačítko „Přihlásit se“.

### *Odesílání zpráv*

Po přihlášení, pokud napíšeme do dolního textového pole zprávu, aktivuje se nám tlačítko „Odeslat“.

Pokud v pravém menu máme vybraného uživatele „Všichni“, bude zpráva odeslána všem. Jinak bude zpráva odeslána pouze konkrétnímu uživateli.



### *Disconnect*

Po stisku tohoto tlačítka se odhlásíme ze serveru. Klient se dostane do výchozího stavu.

### *Ping*

Pomocí stisku tohoto tlačítka lze kontrolovat odezvy vůči ostatním uživatelům.

POZOR! Ping nelze použít na uživatele „Všichni“

### *Refresh*

Pomocí tohoto tlačítka aktualizujeme seznam všech uživatelů po pravé straně klienta.