Оглавление

1. Введение
   1. Цели и задачи
2. Теория
   1. WCF/WPF
   2. Актуальность и перспективы
   3. Структура проекта
3. Практика
   1. Диаграммы
      1. Классов
      2. Состояния
      3. Use/case
      4. Листинг
      5. GUI
4. Реализация
   1. Тестирование

Введение

Цели и задачи

Тема моей курсовой работы – создание WCF(**Windows Communication Foundation)** чата.

Целью моей курсовой работы является написание клиент-серверного приложение с графическим интерфейсом клиента. Взаимодействие клиентов и функционал будет основан на фреймворке WCF. Графический интерфейс реализуется на WPF.

Главные задачи – научиться разрабатывать клиент-серверную архитектуру, использовать фреймворк WCF и научиться проектировать.

Теория

WCF/WPF

**Windows Communication Foundation** (**WCF**) — программный фреймворк, используемый для обмена данными между приложениями, входящий в состав .NET Framework.

WCF делает возможным построение безопасных и надёжных транзакционных систем через упрощённую унифицированную программную модель межплатформенного взаимодействия. Комбинируя функциональность существующих технологий .NET по разработке распределённых приложений (ASP.NET XML Web Services — ASMX, WSE 3.0, .NET Remoting, .NET Enterprise Services и System.Messaging), WCF предоставляет единую инфраструктуру разработки, при умелом применении повышающую производительность и снижающую затраты на создание безопасных, надёжных и транзакционных Web-служб нового поколения.

**Windows Presentation Foundation** (**WPF**) — система для построения клиентских приложений Windows с визуально привлекательными возможностями взаимодействия с пользователем, графическая (презентационная) подсистема в составе .NET Framework (начиная с версии 3.0), использующая язык XAML.

**«Клиент — сервер»** (англ. *client–server*) — вычислительная или сетевая архитектура, в которой задания или сетевая нагрузка распределены между поставщиками услуг, называемыми серверами, и заказчиками услуг, называемыми клиентами. Фактически клиент и сервер — это программное обеспечение. Обычно эти программы расположены на разных вычислительных машинах и взаимодействуют между собой через вычислительную сеть посредством сетевых протоколов, но они могут быть расположены также и на одной машине. Программы-серверы ожидают от клиентских программ запросы и предоставляют им свои ресурсы в виде данных.

Актуальность и перспективы

На самом деле, технологии WCF и WPF довольно таки устаревшие. Официальная документация не обновляется WСF – более 9 лет, WPF –более 4 лет.

В настоящее время, объективно сказать, использование этих технологий совершенно неоправданно. Давно существуют более производительные фреймворки для клиент-серверных приложений.

WCF мало чем отличается от других сетевых фреймворков. Однако весьма проигрывает по производительности, функционалу другим популярным back-end ЯП.

Структура проекта

Программа будет разделена на три части: клиентскую, серверную части и родительский класс с интерфейсами и абстрактными классами.

В родительском классе будут реализованы контракт WCF службы.

Клиентская часть будет добавлять функционал графическому интерфейсу.

Серверная часть сокеты и TCP будет принимать и отправлять сообщения по выбранному зарезервированному ip и порту. Однако для реализации на OC Windows сервер придется запускать от имени администратора.

Приложения клиента и сервера, расширившие интерфейс родительского класса будут реализовывать паттерн синглтон, то есть в программе будет находиться исключительно один экземпляр класса как клиента, так и сервера.

Практика

Диаграммы

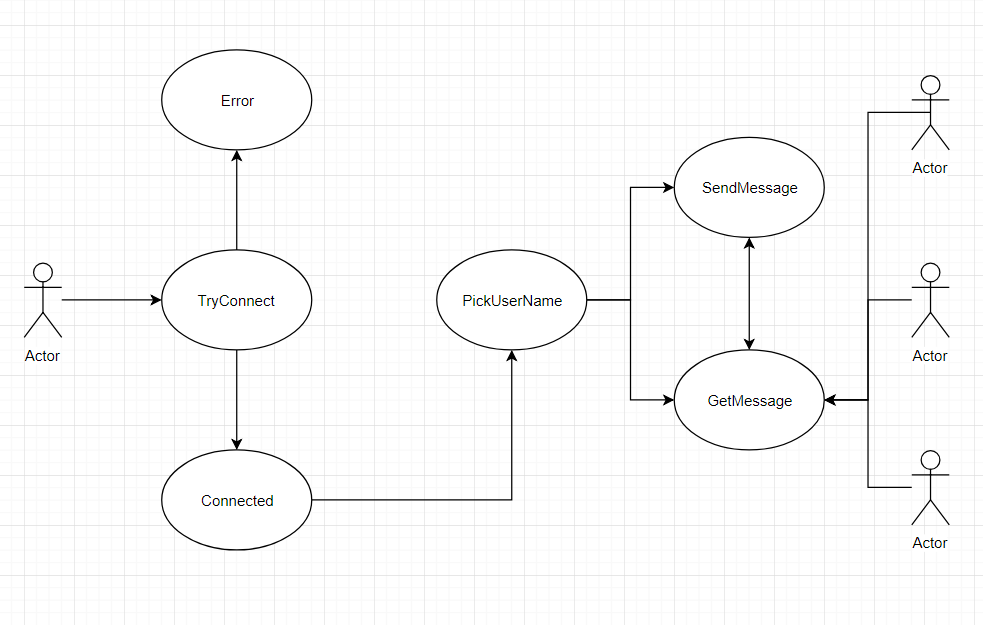


Рис 1.1 Use/Case диаграмма

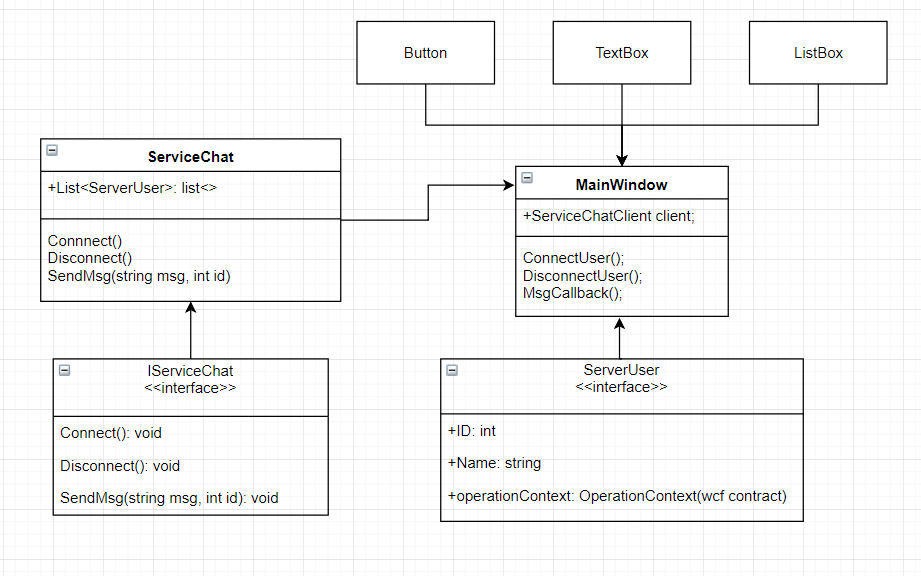


Рис 1.2 диаграмма классов

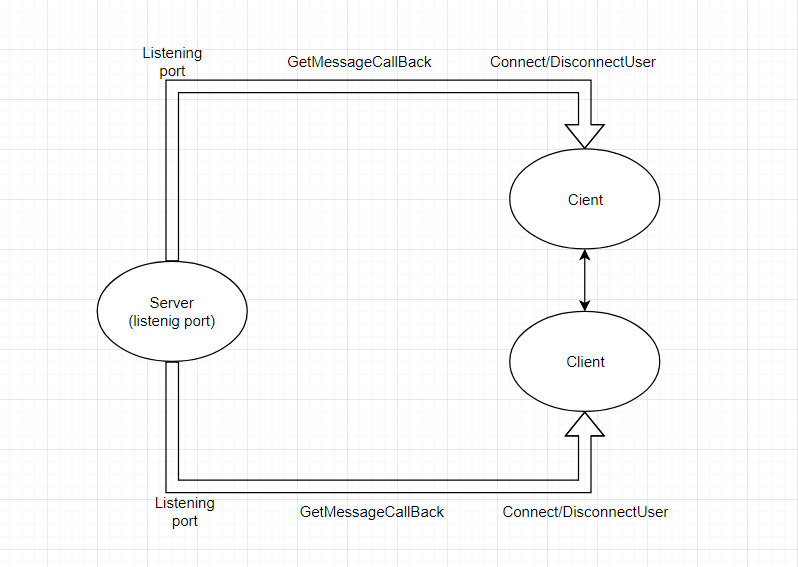


Рис 1.3 диаграмма состояний

* На Use/Case диаграмме рис1.1 можно увидеть общий принцип работы программы. Да и вообще чатов в общем. Пользователь(актор) подключается к серверу, выбирает себе никнейм и начинает сессию. Он получает сообщения от других акторов и может сам отправлять сообщения.
* На диаграмме классов визуализирован раннее описанный способ взаимодействия структур программы. Расширены и переопределены интерфейсы, а gui использует стандартные механизмы button, textbox и listbox.
* На рис1.3 изображена диаграмма состояний. Сервер статично находится в состоянии прослушивания текущего порта. Клиенты в свою очередь способны занять порт и начать обмениваться с сервером сообщениями в многопоточнмо режиме, естественно. Многопоточный режим, стоит упомянуть, контрактом WCF реализуется «из коробки».

Листинг кода

* ***wcf\_chat:***

***Интерфейс IServiceChat():***

1. [ServiceContract(CallbackContract = typeof(
   * + 1. IServerChatCallback))]
2. public interface IServiceChat
3. {
4. [OperationContract] int Connect(string name);
5. [OperationContract] void Disconnect(int id);
6. [OperationContract(IsOneWay = true)] void SendMsg(string msg, int id);
7. }
8. public
9. interface IServerChatCallback
10. {
11. [OperationContract(IsOneWay = true)] void MsgCallback(string msg);
12. }

Здесь я объявляю методы для определения и «заключаю» контракт с службами WCF.

***Интерфейс ServerUser():***

1. public
2. class ServerUser {
3. public
4. int ID {
5. get;
6. set;
7. }
8. public
9. string Name {
10. get;
11. set;
12. }
13. public
14. OperationContext operationContext {
15. get;
16. set;
17. }

}Здесь обозначаю основные поля.

***Класс ServiceChat():IServiceChat() :***

1. List<ServerUser> users = new List<ServerUser>();
2. int nextId = 1;
3. public int Connect(string name)
4. {
5. ServerUser user = new ServerUser() {
6. ID = nextId,
7. Name = name,
8. operationContext = OperationContext.Current
9. };
10. nextId++;
11. SendMsg(": "+user.Name+" подключился",0);
12. Console.WriteLine("{0}, connected {1}",user.Name, DateTime.Now.ToShortTimeString());
13. users.Add(user);
14. return user.ID;
15. }
16. public void Disconnect(int id)
17. {
18. var user = users.FirstOrDefault(i => i.ID == id);
19. if (user!=null)
20. {
21. users.Remove(user);
22. SendMsg(": "+user.Name + " покинул чат",0);
23. Console.WriteLine("{0}, disconnected {1}", user.Name, DateTime.Now.ToShortTimeString());
24. }
25. }
26. public void SendMsg(string msg, int id)
27. {
28. foreach (var item in users)
29. {
30. string answer = DateTime.Now.ToShortTimeString();
31. var user = users.FirstOrDefault(i => i.ID == id);
32. if (user != null)
33. {
34. answer += ": " + user.Name+" ";
35. }
36. answer += msg;
37. item.operationContext.GetCallbackChannel<IServerChatCallback>().MsgCallback(answer);
38. }

Здесь реализую метода и контракт. При подключении и отключении юзера отправляю сообщение всем пользователям, а также логгирую в терминал сервера. Список пользователей реализован списком стандартного контейнера list.

* ***ChatHost:***

***Вход main():***

1. static void Main(string[] args)
2. {
3. using (var host = new ServiceHost(typeof(wcf\_chat.ServiceChat)))
4. {
5. host.Open();
6. Console.WriteLine("host started");
7. Console.ReadLine();
8. }
9. }

Здесь используем реализованный класс и запускаем консоль для логов.

* ***ChatClient:***

***Функционал гуя MainWindow.xaml:***

1. bool isConnected = false;
2. ServiceChatClient client;
3. int ID;
4. public
5. MainWindow() { InitializeComponent(); }
6. private void Window\_Loaded(object sender, RoutedEventArgs e) {}
7. void ConnectUser() {
8. if (!isConnected) {
9. client =
   1. new ServiceChatClient(new System.ServiceModel.InstanceContext(this));
10. ID = client.Connect(tbUserName.Text);
11. tbUserName.IsEnabled = false;
12. bConnDicon.Content = "Disconnect";
13. isConnected = true;
14. }
15. }
16. void DisconnectUser() {
17. if (isConnected) {
18. client.Disconnect(ID);
19. client = null;
20. tbUserName.IsEnabled = true;
21. bConnDicon.Content = "Connect";
22. isConnected = false;
23. }
24. }
25. private void Button\_Click(object sender, RoutedEventArgs e) {
26. if (isConnected) {
27. DisconnectUser();
28. } else {
29. ConnectUser();
30. }
31. }
32. public void MsgCallback(string msg) {
33. lbChat.Items.Add(msg);
34. lbChat.ScrollIntoView(lbChat.Items[lbChat.Items.Count - 1]);
35. }
36. private void Window\_Closing(object sender, System.ComponentModel.CancelEventArgs e) {
37. DisconnectUser();
38. }
39. private void tbMessage\_KeyDown(object sender, KeyEventArgs e) {
40. if (e.Key == Key.Enter) {
41. if (client != null) {
42. client.SendMsg(tbMessage.Text, ID);
43. tbMessage.Text = string.Empty;
44. }
45. }

Для нажатия кнопки использую метод ConnectUser(), для введения текста испоьльзую контейнер string. Отпрвляю по нажатия кнопки. Также вывожу состояние подклюяение.

* ***GUI:***

***MainWindow.xaml:***

**< Window** x:Class="ChatClient.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:ChatClient"

mc:Ignorable="d"

Title="MainWindow" Height="350" Width="525" Loaded="Window\_Loaded" Closing="Window\_Closing"**>**

**<Grid>**

**<Grid.ColumnDefinitions>**

**<ColumnDefinition** Width="112\*"**/>**

**<ColumnDefinition** Width="407\*"**/>**

**</Grid.ColumnDefinitions>**

**<Grid.Background>**

**<LinearGradientBrush>**

**<GradientStop** Color="SpringGreen" Offset="0" **/>**

**</LinearGradientBrush>**

**</Grid.Background>**

**<TextBox** x:Name="tbUserName" HorizontalAlignment="Left" Height="29" Margin="60.2,10,0,0" TextWrapping="Wrap" Text="Никнейм" VerticalAlignment="Top" Width="335" Grid.Column="1"**/>**

**<Button** x:Name="bConnDicon" Background="#0bda51"

Click="Button\_Click" Content="Connect"

HorizontalAlignment="Left" Margin="10,10,0,0"

VerticalAlignment="Top" Width="147"

Height="29" Grid.ColumnSpan="2"**/>**

**<ListBox** x:Name="lbChat" HorizontalAlignment="Left"

Height="181" Margin="10,44,0,0"

VerticalAlignment="Top" Width="497" Grid.ColumnSpan="2" **/>**

**<TextBox** x:Name="tbMessage" HorizontalAlignment="Left"

Height="79" Margin="10,230,0,0"

TextWrapping="Wrap" Text=""

VerticalAlignment="Top" Width="497" KeyDown="tbMessage\_KeyDown" Grid.ColumnSpan="2"**/>**

**</Grid>**

</>

Здесь я создал с помощью WPF примитивный графический интерфейс с интуитивно-понятным доступом. Есть текстовое поле для никнейма, кнопка подключение , текстбокс для непосредственно самого сообщения пользователя и текстовое поле-лог сообщений с датой, временем и именем пользователя, отправившего его.

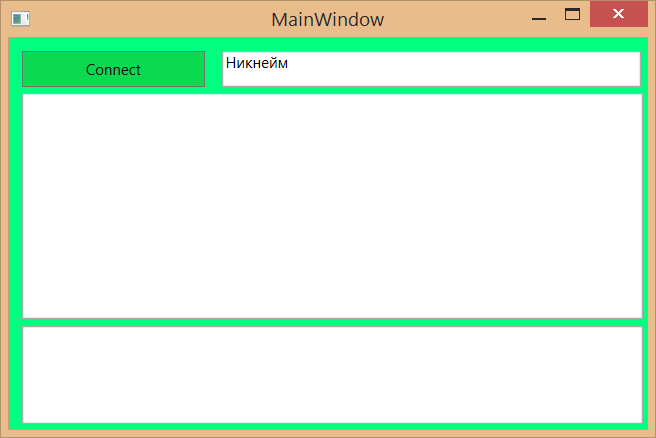
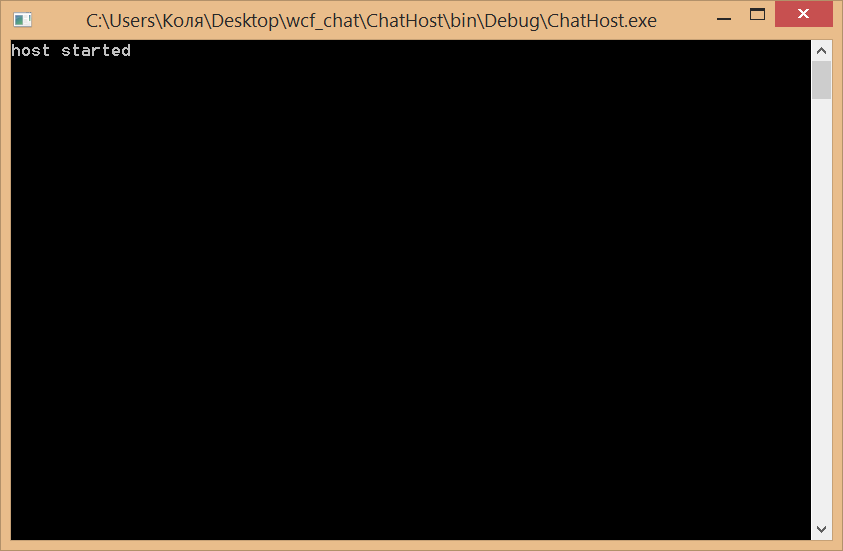


Рис 2.1 GUI

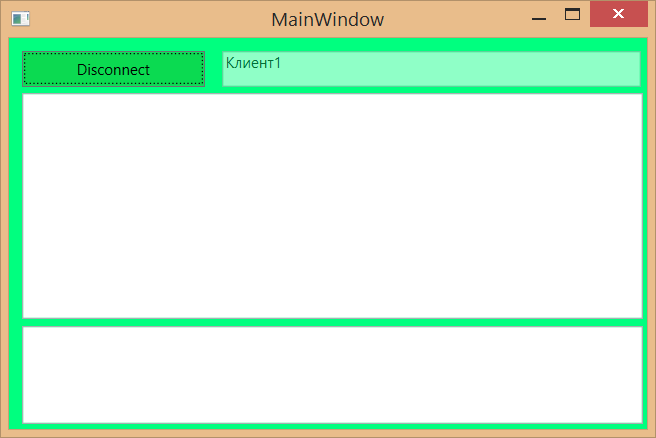
Реализация

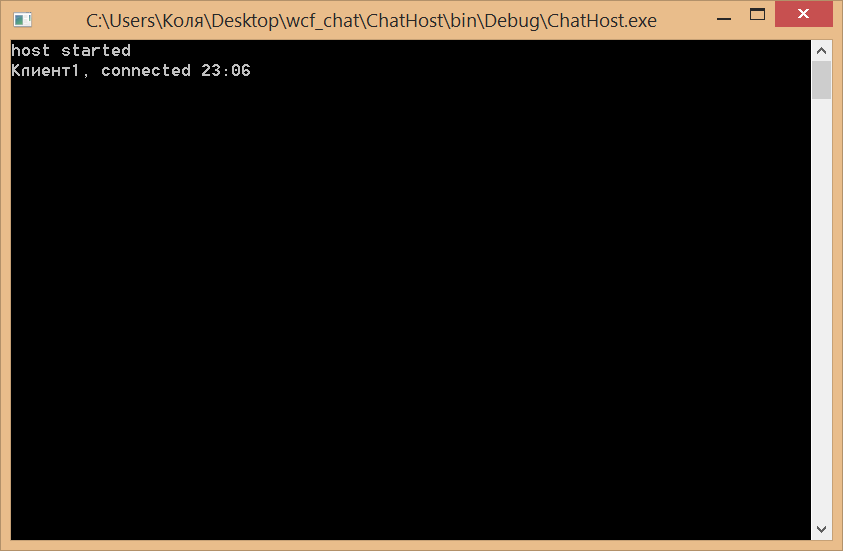
Тесты

1)Запустим сервер от имени администратора

рис 3.1 консоль сервера

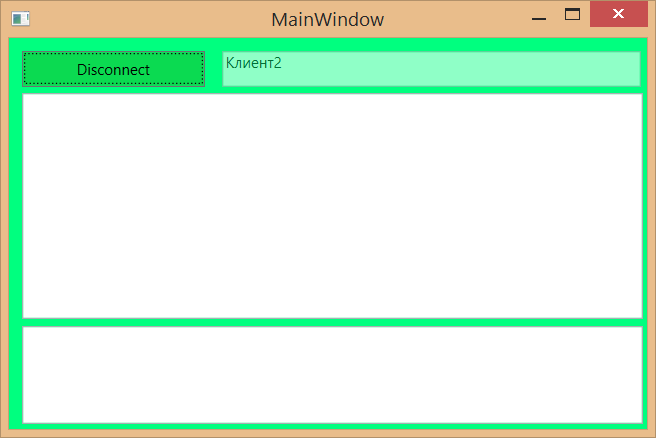
2) запустим первый клиент, назовем его клиент1 и подключимся:

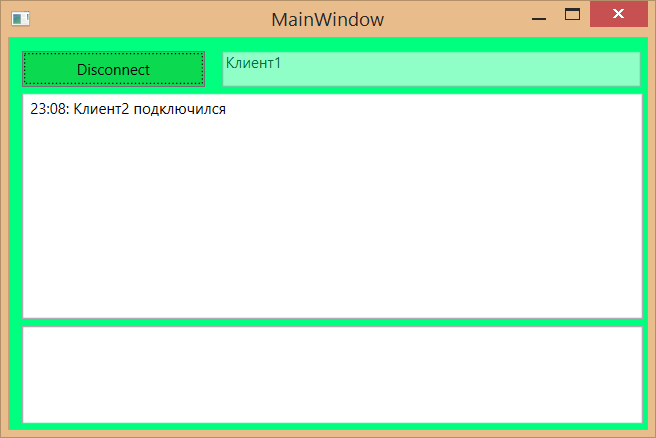
рис 3.2 Клиент1

рис 3.3 лог подключения

Как видим, лог работает, клиент подключился.

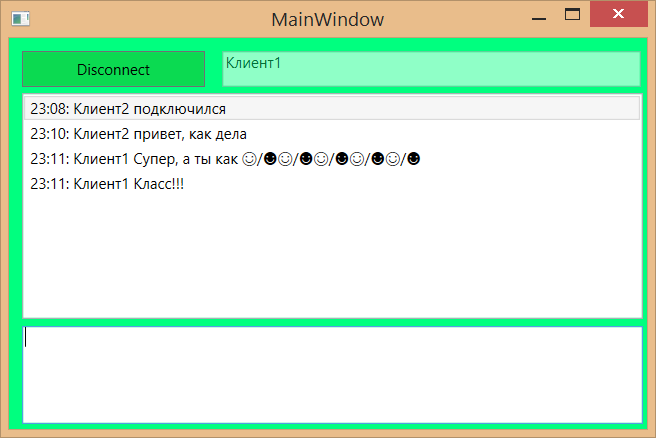
3) Аналогично сделаем со вторым клиентом, назовем его клиент2:

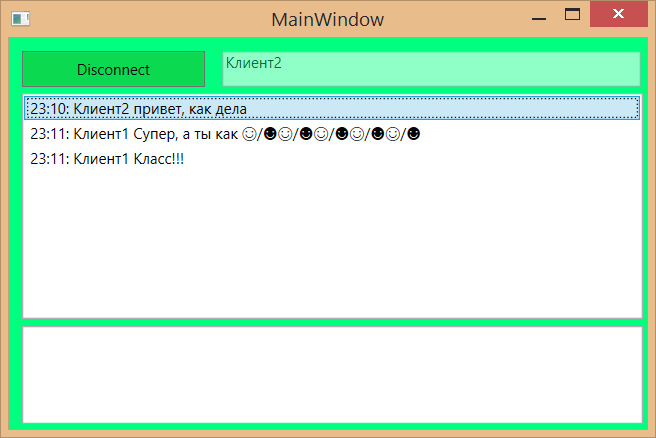
рис3.4 Клиент2

рис 3.5 Отображение в чате

Удачно прошел вывод подключения клиента2 в чат у клиента1.

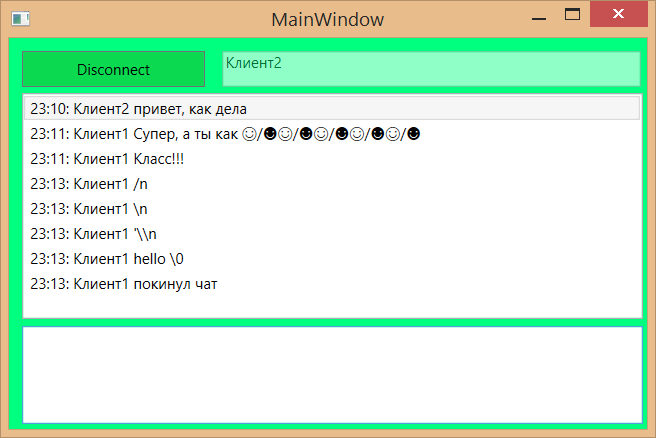
4)Тестируем обмен сообщениями



рис 3.6 Чат

Чат свои функции безукоризненно выполняет. Цель выполнена!

5)Покинем чат

рис.3.7 вывод статуса.

Для клиента2 выведено сообщение, все как и ожидалось.