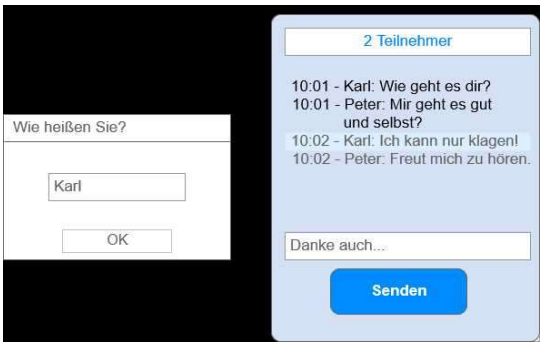
Ein WCF Chat Programm

Aufgabenstellung

Die Clients sind WPF-Fenster (siehe Anhang). Startet man das Programm wird man zunächst nach einem Namen gefragt. Danach öffnet sich das eigentliche Hauptfenster. In diesem kann man unten einen Text eingeben. Beim Klicken auf „Senden“ wird dieser Text und der Name des Nutzers an den Server geschickt. Der Server sendet ihn außerdem mit Zeitstempel an alle Clients und speichert ihn intern in einem Chatverlauf. Beim ersten Anmelden bekommt ein Client außerdem den kompletten Chatverlauf seit dem Neustart des Servers (wir brauchen keine Datenbanken oder irgendwas, nur eine Liste mit Chatbeiträgen). Weiterhin wird in einem Chatfenster auch immer angezeigt wie viele Personen aktuell für den Chat angemeldet sind. Schließt jemand das Programm wird er automatisch abgemeldet und verschwindet aus der Liste.



Beispiel einer Chat Apps

Technologien

- Server: WCF in .NET Framework 4.8
- Client: WPF

Source Code

Das Projekt liegt in Github: https://github.com/xdah031/Vergleich-WCF-Alternativen. Sie enthält zwei Verzeichnisse: WPFChatView für Client und WCFChat für Server.

Anleitung:

- Führen Sie WCFChat.sln Solution aus.
- Starten Sie Chat Dienst im WCFChat Projekt. Einfachsten ist beim rechten Klick auf Chat.svc => In Browser anzeigen
- Starten Sie die WPF Anwendung im WPFChatView Projekt oder im Verzeichnis .\WCFChat\WPFChatView\bin\Debug\WPFChatView.exe, wenn Sie mehrere Clients parallel simulieren wollten.

Chat Server

Um den Server richtig zu erstellen, brauchen Wir in WCF mindesten drei Dinge: *DataContract*, *ServiceContract* und die Methode für *ServiceContract* oder *ServiceBehavior*. Außer dem ist der *CallbackContract* erforderlich, um Daten aus Server zu Client zu senden.

DataContract

Code:

```
[DataContract]
    public class ChatInfos
    {
        string name = "Anonymous";
        string chat = "";
        DateTime time = DateTime.MinValue;

        public ChatInfos() { }

        public ChatInfos(string name, string chat)
        {
            this.name = name;
            this.chat = chat;
        }

        public ChatInfos(string name, string chat, DateTime time)
        {
            this.name = name;
            this.chat = chat;
            this.time = time;
        }

        [DataMember]
        public string Name
        {
            get { return name; }
            set { name = value; }
        }

        [DataMember]
        public string Chat
        {
            get { return chat; }
            set { chat = value; }
        }

        [DataMember]
        public DateTime Time
        {
            get { return time; }
            set { time = value; }
        }
    }
```

In *DataContract* definieren wir den Typ der Nachricht. Sie ist *ChatInfos* gennant. Jede Nachricht hat drei Eigenschaften:

- Name: Name vom Client.
- Chat: Inhalt der Nachricht.
- Time: Zeitpunkt, wann der Server die Nachricht erhält.

ServiceContract

Code:

```
[ServiceContract(CallbackContract = typeof(IChatClient),
                 SessionMode = SessionMode.Required)]
public interface IChat
{
    [OperationContract(IsOneWay = true)]
    void SendChat(string name, string text);

    [OperationContract(IsOneWay = true, IsInitiating = true)]
    void Join(string name);

    [OperationContract(IsOneWay = true)]
    void Refresh();

    [OperationContract(IsOneWay = true, IsTerminating = true)]
    void LogOut();

    [OperationContract(IsOneWay = true)]
    List<ChatInfos> GetChats();
}
```

Zum Entwerfen des *ServiceContract* müssen wir wissen, wie der Client mit dem Server interagieren wird. Wir gehen davon aus, dass *SessionMode* eingeschaltet wird. Zuerst verbindet der Client mit dem Server und beginnt seine Session mithilfe von „*IsInitiating*“. Dann wird die Client-Liste automatisch aktualisiert. Client kann nun Nachricht senden. Zum Schluss endet Client seine Session mithilfe von „*IsTerminating*“.

Außerdem referenziert *ServiceContract* Callback-Schnittstelle, die der Client implementieren wird.

CallbackContract

Code:

```
public interface IChatClient
{
    [OperationContract(IsOneWay = true)]
    void ReceiveChat(ChatInfos chatInfos);

    [OperationContract(IsOneWay = true)]
    void RefreshUserList(List<string> userList);
}
```

Callback-Schnittstelle definiert nur einige Operationen, die auf der Clientseite aufgerufen werden sollen.

ServiceBehavior

Code:

```
[ServiceBehavior(ConcurrencyMode = ConcurrencyMode.Multiple,
                 InstanceContextMode = InstanceContextMode.Single)]
    public class Chat : IChat
    {
        Dictionary<IChatClient, string> users = new Dictionary<IChatClient,
                                                               string>();
        List<ChatInfos> history = new List<ChatInfos>();
        List<string> userList = new List<string>();

        private Chat() { }

        public List<ChatInfos> GetChats()
        {
            return history;
        }

        public void Join(string name)
        {
            var connection =
                    OperationContext.Current.GetCallbackChannel<IChatClient>();
            users[connection] = name;
            userList.Add(name);
        }

        public void LogOut()
        {
            var connection =
                    OperationContext.Current.GetCallbackChannel<IChatClient>();
            userList.Remove(users[connection]);
            users.Remove(connection);
            Refresh();
        }

        public void Refresh()
        {
            foreach (var user in users.Keys)
            {
                user.RefreshUserList(userList);
            }
        }

        public void SendChat(string name, string text)
        {
            var connection =
                    OperationContext.Current.GetCallbackChannel<IChatClient>();
            ChatInfos newChat = new ChatInfos(name, text, DateTime.Now);
            history.Add(newChat);

            if (!users.TryGetValue(connection, out name))
                return;

            foreach (var user in users.Keys)
            {
                user.ReceiveChat(newChat);
            }
        }
    }
```

Hier definieren wir das Verhalten vom *ServiceContract.* Weil wir keine Datenbank verwenden wollten, werden ein Dictionary von Client-Channels und zwei Liste von dem Chatverlauf und Clients erzeugt. Um den aktuellen Channel zu finden und verwenden, erstellen wir eine neue Variable *connection* durch die Methode *„OperationContext.Current.GetCallbackChannel<IChatClient>()“.* Diese *connection* wird durch *Join* zu *Dictionary* hinzugefügt und durch *LogOut* entfernt. Für Broadcast-Nachricht senden wir einfach die Nachricht zu allen Clients in Dictionary *users.*

Web.config

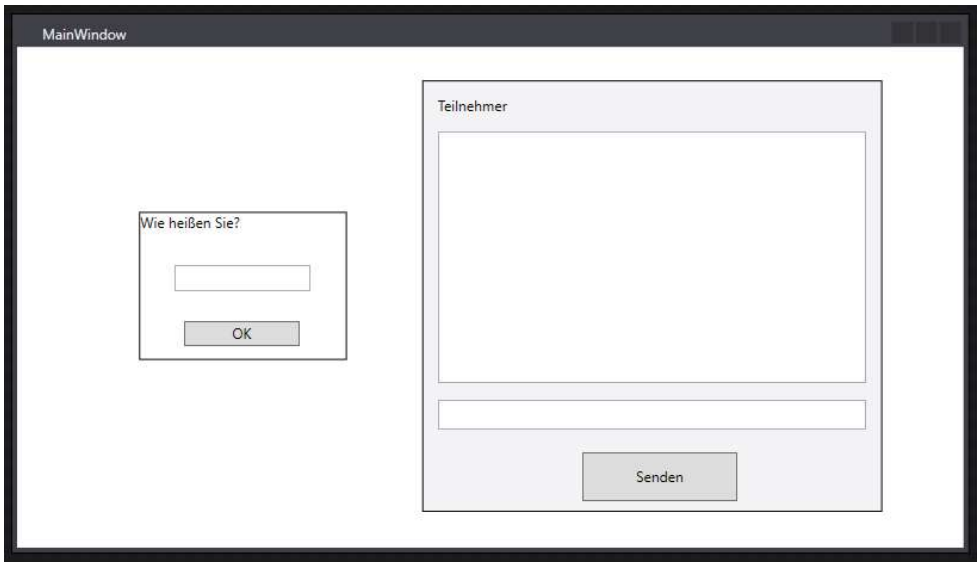
Code:

```
...
    <services>
      <service name="WCFChat.Chat">
        <endpoint address=""
                  binding="wsDualHttpBinding"
                  contract="WCFChat.IChat">
        </endpoint>
      </service>
    </services>
...
```

Wir müssen die *binding*-Eigenschaft auf *wsDualHttpBinding* einstellen. Sie ist eine sichere und interoperable Bindung, die für die Verwendung mit Duplexdienstverträgen konzipiert ist, die es sowohl Diensten als auch Clients ermöglichen, Nachrichten zu senden und zu empfangen. Die leere *address* bedeutet, dass die Adresse vom Service automatisch erstellt werden soll.

Chat Client

Die Oberfläche wird durch WPF-Toolbox erstellt. Das Chatfenster wird angezeigt, nach dem wir den Name-TextBox ausfüllen und „OK“-Button klicken.



User Control

Danach müssen wir den Dienstverweis zum Chatdienst, die wir oben erstellt haben, hinzufügen. Beim rechten Klick auf WPFChatView/Hinzufügen/Dienstverweis… erscheint ein visuelles Werkzeug. Füllen wir die Adresse vom Chatdienst und dann Namespace aus, wird die Verbindung mit Server hergestellt.

Callback Funktionen

Code:

```
    public class MyCallBack : ChatServiceReference.IChatCallback
    {
        public void ReceiveChat(ChatInfos chatInfos)
        {
            Window window = Application.Current.MainWindow;
            (window as MainWindow).chatPanel.Text +=
                                (window as MainWindow).DisplayChat(chatInfos) +
                                                        Environment.NewLine;
        }

        public void RefreshUserList(string[] userList)
        {
            Window window = Application.Current.MainWindow;
            (window as MainWindow).userCount.Text =
                                            userList.Length + " Teilnehmer:";
            foreach (string user in userList)
                (window as MainWindow).userCount.Text += " " + user;
        }
    }
```

Das Verhalten vom *CallbackContract* wird in einer anderen Klasse definiert. Durch die Methode *„ReceiveChat“* bzw. *„RefreshUserList“* erhält Client die Nachricht bzw. neue Client-Liste vom Server, wenn ein anderer Client Methode *SendChat* bzw. *Join* aufruft.

MainWindow

Code:

```
public partial class MainWindow : Window
    {
        string Name = "";
        static InstanceContext context = new InstanceContext(new MyCallBack());
        ChatServiceReference.ChatClient server =
                                new ChatServiceReference.ChatClient(context);
        public MainWindow()
        {
            InitializeComponent();
            ChatView.Visibility = Visibility.Hidden;
        }

        private void Button_Click(object sender, RoutedEventArgs e)
        {
            if (Name == name.Text)
                return;
            else if (Name == "")
            {
                Name = name.Text;
                server.Join(Name);

                ChatView.Visibility = Visibility.Visible;
                var history = server.GetChats();
                foreach (var chat in history)
                    chatPanel.Text += DisplayChat(chat) + Environment.NewLine;
            }
            else
            {
                server.LogOut();
```

```
                Name = name.Text;
                server.Join(Name);
            }

            server.RefreshAsync();
        }

        public string DisplayChat(ChatInfos chat)
        {
            return chat.Time.ToShortTimeString() + " - " +
                                        chat.Name + ": " + chat.Chat;
        }

        private void Button_Click_1(object sender, RoutedEventArgs e)
        {
            if (chatText.Text != "")
                server.SendChat(Name, chatText.Text);
            chatText.Clear();
        }

        void MainWindow_Closing(object sender, CancelEventArgs e)
        {
            if (Name != "")
                server.LogOut();
            server.Close();
        }

        private void OnKeyDownHandler(Object sender, KeyEventArgs e)
        {
            if (e.Key == Key.Enter)
                Button_Click_1(this, new RoutedEventArgs());
        }
    }
```

Diese Klasse enthält alle Hauptfunktionen für die Oberfläche. Zuerst verwenden wir die generierte Clientklasse zum Aufrufen des Diensts. Dann sollen wir die Methoden, die wir in *ServiceContract* definiert haben, in einer richtigen Stelle zuordnen.

Zusammenfassung

Diese einfache Chat Anwendung ist nur ein Beispiel mit dem Ziel, dass man die Verwendung von WCF besser verstehen kann. Deshalb konzentriere ich nicht so stark auf die Ausnahmebehandlung und Optimierung. Weil die Apps auf meiner Lernerfahrung basieren, freue ich mich auf ihre Vorschläge, Korrekturen und Kommentare, wofür die Anwendung verbessert wird.

Vielen Dank für Ihre Geduld, bis hier zu lesen. ☺