Ein WCF Chat Programm

Aufgabenstellung

Die Clients sind WPF-Fenster (siehe Bild). 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 mit MVVM light

Source Code

Das Projekt liegt in Github:

https://github.com/xdah031/Vergleich-WCF-Alternativen/tree/master/WcfChatApplication

Es enthält zwei Verzeichnisse: WpfClient für Client und WcfServer für Server.

Anleitung:

- Führen Sie WcfServer.sln Solution aus.
- Starten Sie Chat Dienst im WcfServer Projekt. Einfachsten ist beim rechten Klick auf ChatService.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 Message
    {
        string user = "Anonymous";
        string text = "";
        DateTime time = DateTime.MinValue;

        public Message() { }

        public Message(string name, string text)
        {
            this.user = name;
            this.text = text;
        }

        public Message(string name, string text, DateTime time)
        {
            this.name = name;
            this.text = text;
            this.time = time;
        }

        [DataMember]
        public string User
        {
            get { return user; }
            set { user = value; }
        }

        [DataMember]
        public string Text
        {
            get { return text; }
            set { text = value; }
        }

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

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

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

ServiceContract

Code:

```
    [ServiceContract(CallbackContract = typeof(IChatCallback),
                     SessionMode = SessionMode.Required)]
    public interface IChatService
    {
        [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<Message> 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 IChatCallback
    {
        [OperationContract(IsOneWay = true)]
        void ReceiveChat(Message Message);

        [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 ChatService : IChatService
    {
        Dictionary<IChatCallback, string> users = new Dictionary<IChatCallback,
                                                                 string>();
        List<Message> history = new List<Message>();
        List<string> userList = new List<string>();

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

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

        public void LogOut()
        {
            var connection =
                    OperationContext.Current.GetCallbackChannel<IChatCallback>();
            if (users.TryGetValue(connection, out string name))
                userList.Remove(name);
            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<IChatCallback>();
            Message newChat = new Message(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<IChatCallback>()“.* 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

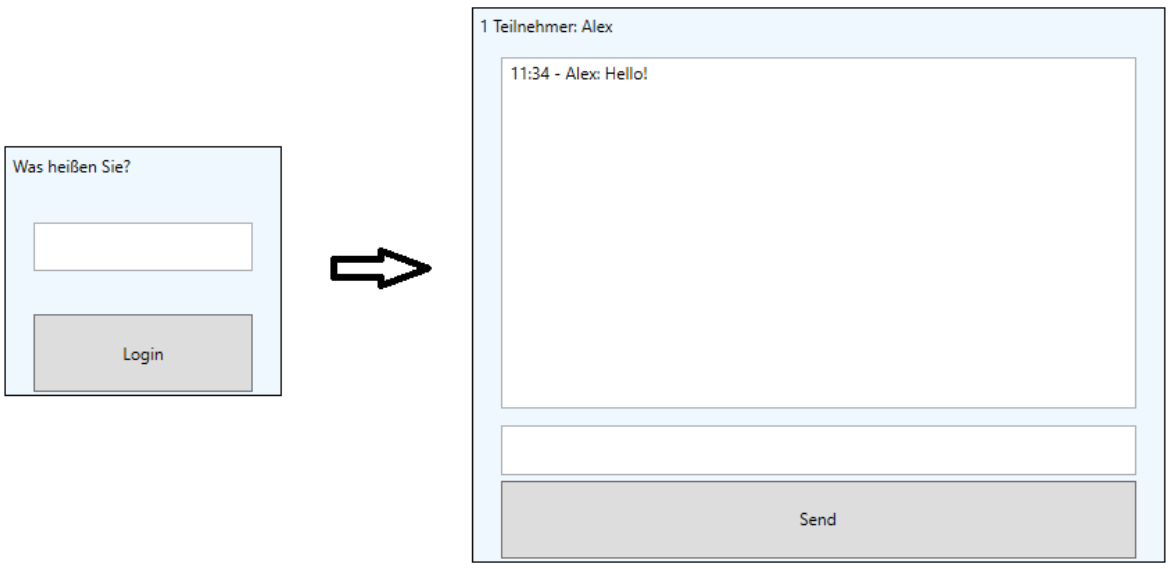
Code:

```
...
    <services>
      <service name="WcfServer.ChatService">
        <endpoint address=""
                  binding="wsDualHttpBinding"
                  contract=" WcfServer.IChatService">
        </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 Application-Oberfläche wird als MainWindow genannt. Sie besteht aus zwei UserControls, LoginUC und ChatViewUC. Das Chatfenster wird angezeigt, nach dem wir den Name-TextBox ausfüllen und „Login“-Button klicken.



User Control

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

ViewModel

Durch MVVM light werden zwei Klasse erzeugt. *MainViewModel* Klasse enthält alle Hauptfunktionen für die Oberfläche, und ViewModelLocator besitzt die Verbindung mit der Anwendung.

Verbindung mit Server

Code:

```
    static InstanceContext context = new InstanceContext(new ChatCallBack());
    ChatServiceClient server = new ChatServiceClient(context);
```

Zuerst verwenden wir die generierte Clientklasse zum Aufrufen des Diensts. Wenn wir die Methoden vom Server aufrufen wollten, benutzen wir „server.‘*Methode_Name*‘(‚*Parameter*‘)“ (z.B.: server.Join(Username).

Initialisierung

Code:

```
        public MainViewModel()
        {
            LoginVisibility = Visibility.Visible;
            ChatViewVisibility = Visibility.Collapsed;

            LoginCommand = new RelayCommand(LoginMethod);
            SendCommand = new RelayCommand(SendMethod);

            if (IsInDesignMode)
            {
                WindowTitle = "Chat Application (Design)";
            }
            else
            {
                WindowTitle = "Chat Application";
            }

            Application.Current.MainWindow.Closing +=
                                new CancelEventHandler(MainWindow_Closing);
        }
```

Amfang wird das Login-Fenster angezeigt und ChatView-Fenster hingegen versteckt. Dann werden die Funktionalitäten zu Login-, Send-Button und Closing-Event hinzugefügt. Der Client wird beim Fenster-schließen ausgeloggt.

Login- und Send-Funktion

Code:

```
        public void LoginMethod()
        {
            if (!string.IsNullOrEmpty(Username))
            {
                LoginVisibility = Visibility.Collapsed;
                ChatViewVisibility = Visibility.Visible;

                server.Join(Username);
                var history = server.GetChats();

                Messages = new ObservableCollection<Message>(history);

                server.RefreshAsync();
            }
        }

        public void SendMethod()
        {
            if (!string.IsNullOrEmpty(ChatText))
                server.SendChat(Username, ChatText);
            ChatText = string.Empty;
        }
```

In dieser Login-Funktion wird der *Username* in Server angemeldet und der Client bekommt dadurch die Chat-Historie und User-Liste. Mit *SendMethod()* sendet der Client die Nachricht am Server. Server wird danach diese Nachricht an allen anderen Client durch *Callback* weiterleiten.

PropertyChanged.Fody

Fody ist ein *Nuget*-Paket, mit dem wir *Event* für *PropertyChanged* nicht selbst definieren müssen. Die Event-Handler werden automatisch in Backend erzeugt, damit der Code sauber und sichtbar wird.

Code:

```
        public string WindowTitle { get; set; }
```

Anstatt:

```
        string windowTitle;
        public string WindowTitle
        {
            get { return windowTitle; }
            set
            {
                windowTitle = value;
                RaisePropertyChanged("WindowTitle");
            }

        }
```

IDataErrorInfo

IDataErrorInfo ist eine Schnittstelle, mit der die Errors behandelt werden können. In diesem Beispiel wird eine einfache Check-Methode, indem der Username geprüft wird, ob er leer ist.

Code:

```
        public string Error => string.Empty;

        public string this[string propertyName]
        {
            get
            {
                switch (propertyName)
                {
                    case nameof(Username):
                        if (Username == "")
                            return "Username is required!";
                        break;
                }
                return string.Empty;
            }
        }
```

Callback Funktionen

Code:

```
    public class ChatCallBack : ChatServiceReference.IChatServiceCallback
    {
        public void ReceiveChat(Message message)
        {
            var main = ServiceLocator.Current.GetInstance<MainViewModel>();
            main.Messages.Add(message);
        }

        public void RefreshUserList(string[] userList)
        {
            var main = ServiceLocator.Current.GetInstance<MainViewModel>();
            string users = userList.Length + " Teilnehmer:";
            foreach (string user in userList)
                users += " " + user;
            main.Userlist = users;
        }
    }
```

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.

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.