29.3.2015

Luca Tirabassi, Alex Theijs

Modulprüfung M120

Dokumentation und Eportfolio zum Chatprogramm

Inhaltsverzeichnis

[1 Aufgabenanalyse 2](#_Toc415435973)

[1.1 Beschreibung des Problems / System 2](#_Toc415435974)

[1.2 Use-Cases für zentrale Aufgaben 2](#_Toc415435975)

[2 Benutzeranalyse 2](#_Toc415435976)

[3 Ergonomieanalyse 2](#_Toc415435977)

[3.1 Grundsätze 2](#_Toc415435978)

[3.2 Maskenlayout und Gruppierung der Information 4](#_Toc415435979)

[4 Eportfolio 5](#_Toc415435980)

[4.1 Was wollen wir lernen 5](#_Toc415435981)

[4.3 Wieso wollen wir das lernen 5](#_Toc415435982)

[4.4 Wie haben wir es gelernt 5](#_Toc415435983)

[4.5 Fazit 8](#_Toc415435984)

[5 Testing 9](#_Toc415435985)

# 1 Aufgabenanalyse

## 1.1 Beschreibung des Problems / System

Unser gewähltes Betriebssystem ist Windows. Weil wir uns für ein Chatprogramm entschieden haben, soll es dem Client möglich sein mit anderen Clients über einen Server miteinander zu kommunizieren. Auch der Server sollte ein Microsoft Windows Betriebssystem haben, da wir mit .NET arbeiten. Des Weiteren sollte eine Internetverbindung vorhanden sein, oder zumindest ein lauffähiges Netzwerk, damit die verschiedenen Clients miteinander kommunizieren können.

## 1.2 Use-Cases für zentrale Aufgaben

Folgende Usecases haben wir ausgearbeitet:

1. User startet Programm
2. User gibt in Textfeld seinen Namen ein
3. User loggt sich mit Login-Button ein
4. User loggt sich mit Enter-Button ein
5. User kann nicht einen Namen länger wie 12 Charaktere eingeben oder mit Sonderzeichen und Nummern
6. User bekommt Fehlermeldung bei Falscheingabe des Nicknames
7. User wählt Chatpartner aus
8. User wählt Chattextbox aus
9. User schreibt in TextBox
10. User sendet geschriebene Nachricht mit Send-Button
11. User sendet geschriebene Nachricht mit Enter-Button
12. User sieht geschriebene Antwort
13. User schliesst Applikation

# 2 Benutzeranalyse

Unsere Zielgruppe sind unerfahrene Computerbenutzer, welche einfach und schnell mit einer kurzen Installation und wenigen Klicks miteinander einfach und schnell kommunizieren wollen.

# 3 Ergonomieanalyse

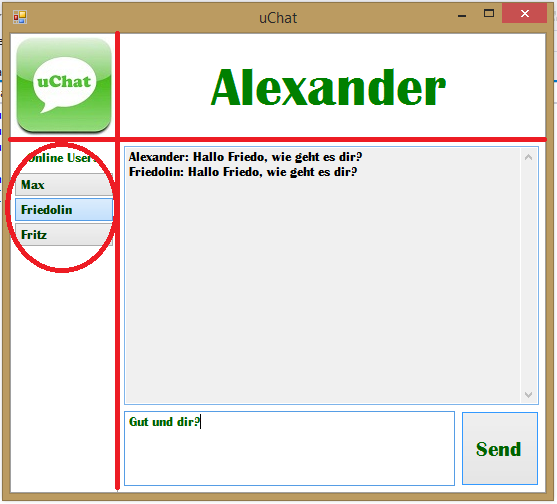
## 3.1 Grundsätze

* Das Chatprogramm soll den Benutzer nicht überfordern und nur das können was wirklich von Ihm erwartet wird.
* Das Programm sollte möglichst selbsterklärend sein 🡪 Das bedeutet, dass der Benutzer ohne jegliches lesen einer Dokumentation mit dem Programm umgehen kann.
* Unser Programm soll ähnlich funktional und Benutzeroberflächen technisch gestaltet sein wie bereits vorhandene Produkte, wie zb.: Skype, ICQ, MSN, Viber
* Dem Benutzer soll es nicht möglich sein falsche Eingaben zu tätigen. Falls dies doch passieren sollte, soll dem Benutzer angezeigt werden, was er falsch gemacht hat.
* Die Bedienbarkeit des Programms soll schnell und einfach lernbar und ausführbar sein

## 3.2 Maskenlayout und Gruppierung der Information

Für unser UI haben wir ein Maskenlayout und eine Gruppierung der Information gewählt.

Dies ist einerseits sichtbar durch das gewählte Tabellenlayout. Andererseits durch die Gruppierung der Benutzer.

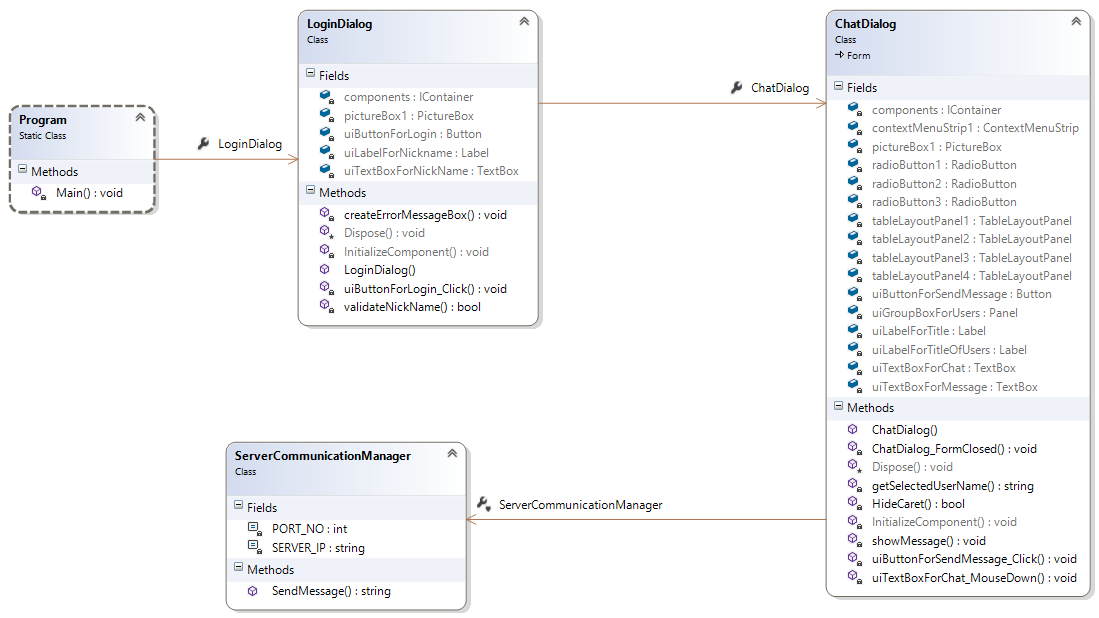


## 3.3 Farben und Kontrast

Wir haben für die Farben Hellgrau, Schwarz, Grün und Weiss genommen. So sieht es modern nach Flat design aus und ist trotzdem für Farbenblinde gut erkennbar. Dies bedeutet es ist Barrierefrei.

# 4 Design

## 4.1 Klassendiagramm



## 4.2 Loginalgorythmus

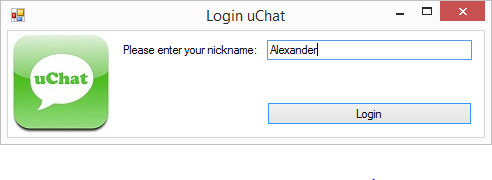


## 4.3 Nachrichtensendealgorythmus

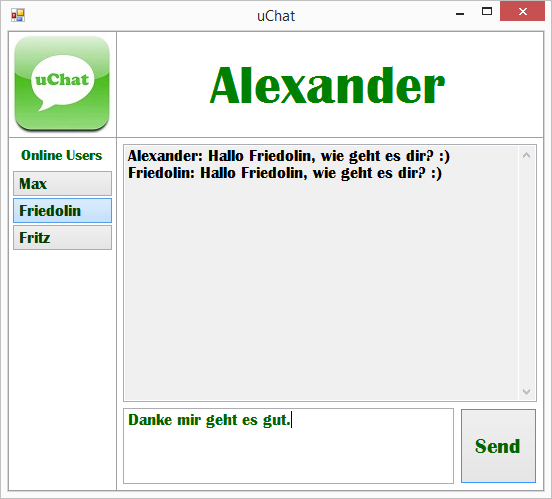


## 4.4 Dialoge

### 4.4.1 lOGIN dIALOG



### 4.4.2 cHAT dIALOG



# 5 Eportfolio

## 5.1 Was wollen wir lernen

Eine Server-Client Kommunikation über Sockets in C#.

## 5.3 Wieso wollen wir das lernen

Da wir uns beide stark für Spieleprogrammierung interessieren, dachten wir wäre es am besten, wenn wir uns in diesem Projekt intensiv mit der Server-Client Kommunikation befassen.  
Dieses Wissen sollte uns als Grundlage für zukünftige Projekte dienen.

## 5.4 Wie haben wir es gelernt

In der Dokumentation von Microsoft sind gute Beispiele für C# Socket Programmierungen vorhanden.  
Auch in den Java-Dokumentationen konnten wir andere Beispiele in einer ähnlichen Sprache vergleichen.  
Durch gewisse Code-Snippets und mit „herumspielen“ und korrigieren von Fehlern und veraltetem Code, kommt man schnell zum gewünschten Ergebnis.

Dies ist das CodeSnippet anhand welchem wir es gelernt haben:

using System;

using System.Net;

using System.Net.Sockets;

using System.Threading;

using System.Text;

// State object for receiving data from remote device.

public class StateObject {

// Client socket.

public Socket workSocket = null;

// Size of receive buffer.

public const int BufferSize = 256;

// Receive buffer.

public byte[] buffer = new byte[BufferSize];

// Received data string.

public StringBuilder sb = new StringBuilder();

}

public class AsynchronousClient {

// The port number for the remote device.

private const int port = 11000;

// ManualResetEvent instances signal completion.

private static ManualResetEvent connectDone =

new ManualResetEvent(false);

private static ManualResetEvent sendDone =

new ManualResetEvent(false);

private static ManualResetEvent receiveDone =

new ManualResetEvent(false);

// The response from the remote device.

private static String response = String.Empty;

private static void StartClient() {

// Connect to a remote device.

try {

// Establish the remote endpoint for the socket.

// The name of the

// remote device is "host.contoso.com".

IPHostEntry ipHostInfo = Dns.Resolve("host.contoso.com");

IPAddress ipAddress = ipHostInfo.AddressList[0];

IPEndPoint remoteEP = new IPEndPoint(ipAddress, port);

// Create a TCP/IP socket.

Socket client = new Socket(AddressFamily.InterNetwork,

SocketType.Stream, ProtocolType.Tcp);

// Connect to the remote endpoint.

client.BeginConnect( remoteEP,

new AsyncCallback(ConnectCallback), client);

connectDone.WaitOne();

// Send test data to the remote device.

Send(client,"This is a test<EOF>");

sendDone.WaitOne();

// Receive the response from the remote device.

Receive(client);

receiveDone.WaitOne();

// Write the response to the console.

Console.WriteLine("Response received : {0}", response);

// Release the socket.

client.Shutdown(SocketShutdown.Both);

client.Close();

} catch (Exception e) {

Console.WriteLine(e.ToString());

}

}

private static void ConnectCallback(IAsyncResult ar) {

try {

// Retrieve the socket from the state object.

Socket client = (Socket) ar.AsyncState;

// Complete the connection.

client.EndConnect(ar);

Console.WriteLine("Socket connected to {0}",

client.RemoteEndPoint.ToString());

// Signal that the connection has been made.

connectDone.Set();

} catch (Exception e) {

Console.WriteLine(e.ToString());

}

}

private static void Receive(Socket client) {

try {

// Create the state object.

StateObject state = new StateObject();

state.workSocket = client;

// Begin receiving the data from the remote device.

client.BeginReceive( state.buffer, 0, StateObject.BufferSize, 0,

new AsyncCallback(ReceiveCallback), state);

} catch (Exception e) {

Console.WriteLine(e.ToString());

}

}

private static void ReceiveCallback( IAsyncResult ar ) {

try {

// Retrieve the state object and the client socket

// from the asynchronous state object.

StateObject state = (StateObject) ar.AsyncState;

Socket client = state.workSocket;

// Read data from the remote device.

int bytesRead = client.EndReceive(ar);

if (bytesRead > 0) {

// There might be more data, so store the data received so far.

state.sb.Append(Encoding.ASCII.GetString(state.buffer,0,bytesRead));

// Get the rest of the data.

client.BeginReceive(state.buffer,0,StateObject.BufferSize,0,

new AsyncCallback(ReceiveCallback), state);

} else {

// All the data has arrived; put it in response.

if (state.sb.Length > 1) {

response = state.sb.ToString();

}

// Signal that all bytes have been received.

receiveDone.Set();

}

} catch (Exception e) {

Console.WriteLine(e.ToString());

}

}

private static void Send(Socket client, String data) {

// Convert the string data to byte data using ASCII encoding.

byte[] byteData = Encoding.ASCII.GetBytes(data);

// Begin sending the data to the remote device.

client.BeginSend(byteData, 0, byteData.Length, 0,

new AsyncCallback(SendCallback), client);

}

private static void SendCallback(IAsyncResult ar) {

try {

// Retrieve the socket from the state object.

Socket client = (Socket) ar.AsyncState;

// Complete sending the data to the remote device.

int bytesSent = client.EndSend(ar);

Console.WriteLine("Sent {0} bytes to server.", bytesSent);

// Signal that all bytes have been sent.

sendDone.Set();

} catch (Exception e) {

Console.WriteLine(e.ToString());

}

}

public static int Main(String[] args) {

StartClient();

return 0;

}

}

## 5.5 Fazit

Es gibt viele verschiedene Vorgehensweisen, wie man einen Server oder einen Client programmiert. Je nach Anforderung müssen gewisse Punkte behandelt worden sein (implementiert), damit der Server oder der Client auch richtig kommunizieren können.

Für unser Beispiel haben wir einen ganz simplen Server programmiert, der aus relativ wenigen Zeilen besteht.  
Wie die Kommunikation miteinander funktioniert hat man nach dem Betrachten des Codes schnell verstanden.

# 6 Testing

|  |  |  |
| --- | --- | --- |
| Testfallbeschreibung | Erwartetes Resultat | Resultat |
| User startet Programm | Login Fenster öffnet sich | true |
| User gibt in Textfeld seinen Namen ein | Textfeld ist mit gewünschtem Nickname befüllt | true |
| User loggt sich mit Login-Button ein | Das Chatfenster öffnet sich | true |
| User loggt sich mit Enter-Button ein | Das Chatfenster öffnet sich | true |
| User gibt Nicknamen mit mehr als 12 Charakteren ein | Fehlermeldung mit Information öffnet sich | true |
| User gibt Nicknamen mit Sonderzeichen oder Zahlen ein | Fehlermeldung mit Information öffnet sich | true |
| User wählt Chatpartner aus | Ausgewählter Chatpartner wird markiert | true |
| User wählt Chattextbox aus | Cursor wird in Nachrichtentextbox angezeigt | true |
| User schreibt in TextBox | Eingabe wird in Nachrichtentextbox angezeigt | true |
| User sendet geschriebene Nachricht mit SendButton | Nachrichtenbox wird geleert. Antwort des Server auf erfolgreiches Senden wird zurück gesendet und in der Chatanzeige angezeigt | true |
| User sendet geschriebene Nachricht mit EnterButton | Nachrichtenbox wird geleert. Antwort des Server auf erfolgreiches Senden wird zurück gesendet und in der Chatanzeige angezeigt | true |
| User sendet leere Nachricht | Nichts passiert | true |
| User sendet Menge von Nachrichten die den Platz des Bildschirmes komplett ausfüllen | Eine Scrollbar erscheint welche das scrollen ermöglicht | true |
| User gibt Nachricht Scrollbar ein | Scrollbar bewegt sich automatisch zum letzten Punkt | true |
| User klickt auf Chatverlauf | Kein Cursor wird angezeigt | true |
| User bekommt Antwort vom Chatpartner | Der richtige Chatpartnername wird im Chatverlauf angezeigt | true |
| User resizes dialog | Resizing doesn’t break the layout | true |
| User will chatverlauf lesen | Chatschrift ist lesbar | false. Die Schriftart oder Größe ist unleserlich |
| User will applikation mit exit button schliessen | Applikation schliesst sich | False. Applikation schliesst sich nicht |
| User schliesst Applikation mit x auf der oberen rechten Ecke | Apllikation schliesst sich | true |