PROJEKT_CHAT1_SERVER – DOKUMENTACJA

Autor: *Krzysztof Rudzki*

Język: **C#**

Połączenie: **TCP (sockets, threads)**

Opis serwera chat, który łączy klientów chat do wspólnej wymiany informacji.

Repozytorium: https://github.com/krzysieker/PROJEKT_CHAT1_SERVER

Spis treści

1. UŻYTY PROTOKÓŁ

Do połączenia użyto protokołu TCP na socketach, z wykorzystaniem wątków.

```
addressIP = IPAddress.Parse(mainForm.GetIPString());
iPEndPoint = new IPEndPoint(addressIP, mainForm.GetPortNumber());

socketServer = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
```

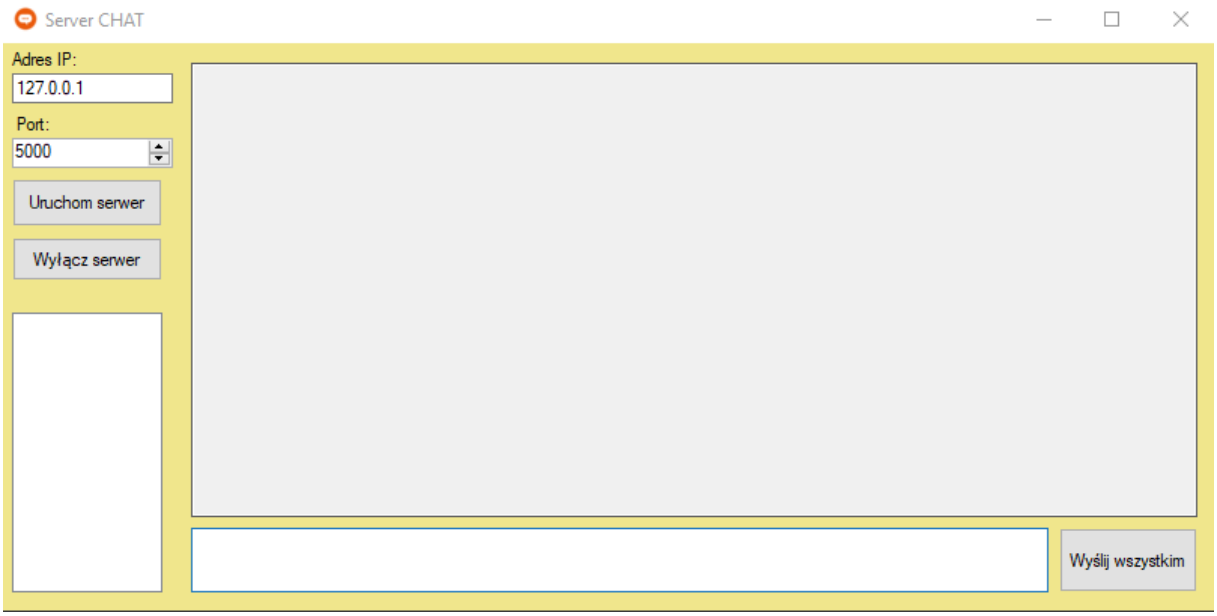
Maksymalna liczba klientów: 100

socketServer.Listen(100);

2. OKNO APLIKACJI

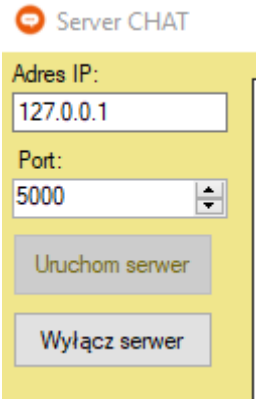
Po uruchomieniu serwera pojawi się główne okno aplikacji. Z racji iż jest to wersja 1.0 to jest to jedyne okno dostępne na ten czas. W przyszłości można rozbudowywać projekt np. dodając formularze logowania czy zaawansowane zarządzanie klientami.

Okno po uruchomieniu wygląda następująco:



- **TextBox: Adres IP:** służy do wprowadzenia adresu IP naszego serwera. Domyślnie ustawiono wartość 127.0.0.1, gdyż klienci byli uruchamiani na tym samym urządzeniu co serwer. W przypadku lokalnej sieci można ustawić IP serwera w tej sieci.
- **TextBox: Port:** analogicznie do adres IP. Domyślnie zarówno na serwerze jak i kliencie ustawiono jego wartość na 5000,
- **Button: Uruchom serwer**: tak jak nazwa wskazuje służy do uruchomienia serwera.

  Po uruchomieniu serwera button : Uruchom serwer zostaje zablokowany.



Kod programu odpowiadający za tą funkcję. Ponieważ funkcje głównego formularza wywoływane są w głównej klasie Program.cs, użyto delegatów.

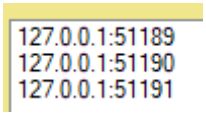
```
delegate void VoidBool(bool bo);

public void SetStartEnabled(bool enabled)
{
    if (this.buttonStart.InvokeRequired)
    {
        VoidBool sse = SetStartEnabled;
        this.textLog.Invoke(sse, enabled);
    }
    else
    {
        this.buttonStart.Enabled = enabled;
    }
}
```

- **Button: Wyłącz serwer:** tak jak nazwa wskazuje służy do uruchomienia serwera. Analogicznie jak w przypadku „Uruchom serwer”

```
public void SetStopEnabled(bool enabled)
{
    if (this.buttonStop.InvokeRequired)
    {
        VoidBool sse = SetStopEnabled;
        this.textLog.Invoke(sse, enabled);
    }
    else
    {
        this.buttonStop.Enabled = enabled;
    }
}
```

- **ListBox (pod buttonami):** Wyświetla IPEndPoint (adres IP + port) aktualnie połączonych z serwerem klientów.

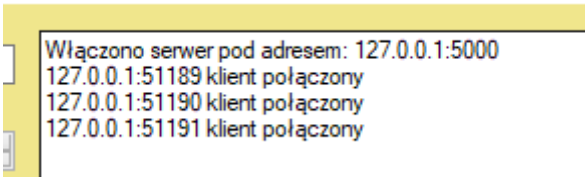
  Dla przykładu:



  Funkcje odpowiedzialne za dodawanie i usuwanie klientów z listy:

```
        public void UsersListAddItem(string str)
        {
            if (this.listBoxUsers.InvokeRequired)
            {
                VoidString listAddItem = UsersListAddItem;
                this.textLog.Invoke(listAddItem, str);
            }
            else
            {
                this.listBoxUsers.Items.Add(str);
            }
        }

        public void UserListRemoveItem(string str)
        {
            if (this.listBoxUsers.InvokeRequired)
            {
                VoidString listRemoveItem = UserListRemoveItem;
                this.textLog.Invoke(listRemoveItem, str);
            }
            else
            {
                this.listBoxUsers.Items.Remove(str);
            }
        }
```

- **TextBox: Logs** (największy): Wyświetla wszelkie komunikaty, wiadomości,



- **TextBox: Message** (na dole): Służy do wprowadzania tekstu do wysłania.

  *Ograniczono liczbę znaków do 1024, gdyż taki bufor danych został zadeklarowany.

- **Button: Wyślij wszystkim:** Służy do wysłania wiadomości z textBox: Message do wszystkich podłączonych klientów.

3. POŁĄCZENIE

Tak jak wspomniano przy omawianiu protokołu użyto TCP na socketach.

```
        static void StartServer(object sender, EventArgs e)
        {
            addressIP = IPAddress.Parse(mainForm.GetIPString());
            iPEndPoint = new IPEndPoint(addressIP, mainForm.GetPortNumber());

            socketServer = new Socket(AddressFamily.InterNetwork, SocketType.Stream,
            ProtocolType.Tcp);

            try
            {
                socketServer.Bind(iPEndPoint);
                socketServer.Listen(100);
                saveLogs.WriteLine("----------------------------");
                PrintlnAndSave($"Włączono serwer pod adresem: {iPEndPoint}");
                mainForm.SetStartEnabled(false);
                mainForm.SetStopEnabled(true);

                Thread thread = new Thread(Listen);
                thread.IsBackground = true;
                thread.Start(socketServer);
            }
            catch (Exception ex)
            {
                PrintlnAndSave($"Błąd: {ex.Message}");
            }
        }
```

4. SZYFROWANIE WIADOMOŚCI

Za szyfrowanie wiadomości odpowiada algorytm BASE64. Jest to podstawowy i mało bezpieczny algorytm ale przynajmniej dane nie są przesyłane plaintextem.

Stworzono klasę Cryptography.cs, która może zostać w przyszłości rozwinięta o bardziej zaawansowane rozwiązania.

```
class Crypto
    {
        public string Base64Encode(string strBeforeBase64)
        {
            var data = Encoding.ASCII.GetBytes(strBeforeBase64);
            var base64 = Convert.ToBase64String(data);
            return base64;
        }

        public string Base64Decode(string encodedDataByBase)
        {
            var decoded = Convert.FromBase64String(encodedDataByBase);
            var result = Encoding.ASCII.GetString(decoded);
            return result;
        }
    }
```

W programie wykorzystano metody klasy, np.

```
string encodedStr = Encoding.UTF8.GetString(buffor, 0, len);
                    string str = crypto.Base64Decode(encodedStr);

                    PrintlnAndSave($"{pointClient}: {str}");

                    foreach (Socket s in clientSockets.Values)
                    {
                        byte[] sendBytes =
Encoding.UTF8.GetBytes(crypto.Base64Encode($"{pointClient}: {str}"));
                        s.Send(sendBytes);
                    }
```

5. Nasłuchiwanie klientów

Za nasłuchiwanie chcących połączyć się klientów odpowiada metoda:

```
        static void Listen(object obj)
        {
            Socket socketServer = obj as Socket;
            while (true)
            {
                try
                {
                    // Czekanie na połączenie
                    Socket socketClient = socketServer.Accept();

                    // Pozyskiwanie adresu IP
                    string pointClient = socketClient.RemoteEndPoint.ToString();
                    PrintlnAndSave($"{pointClient} klient połączony");

                    clientSockets.Add(pointClient, socketClient);
                    mainForm.UsersListAddItem(pointClient);

                    // Otwarcie nowego wątku
                    Thread thread = new Thread(Receive);
                    thread.IsBackground = true;
                    thread.Start(socketClient);
                }
                catch (Exception ex)
                {
                    PrintlnAndSave($"Błąd: {ex.Message}");
                    break;
                }
            }
        }
```

6. ODBIERANIE WIADOMOŚCI I PRZEKAZYWANIE DO INNYCH KLIENTÓW

Poniżej znajduje się listing odpowiedzialny za odbieranie wiadomości i przekazywanie do innych klientów.

- Na odbierane dane ustawiono bufor 1024 bajty, który powinien wystarczyć na przesłanie nawet dłuższych wiadomości,
- Aby usprawnić komunikację pomija się bajty puste,
- Otrzymana wiadomość jest decodowana z powrotem na zrozumiały dla użytkownika ciąg znaków. Następnie jest wysyłana dalej razem z IP i portem danego użytkownika.
- W przypadku błędów klient jest rozłączany (dla bezpieczeństwa).

```
static void Receive(object obj)
        {
            Socket socketClient = obj as Socket;
            string pointClient = socketClient.RemoteEndPoint.ToString();
            while (true)
            {
                try
                {
                    // Uzyskanie wysłanego kontenera wiaodmosci
                    byte[] buffor = new byte[1024];
                    int len = socketClient.Receive(buffor);

                    // pomijanie bajtów pustych
                    if (len == 0)
                    {
                        break;
                    }

                    string encodedStr = Encoding.UTF8.GetString(buffor, 0, len);
                    string str = crypto.Base64Decode(encodedStr);

                    PrintlnAndSave($"{pointClient}: {str}");

                    foreach (Socket s in clientSockets.Values)
                    {
                        byte[] sendBytes =
Encoding.UTF8.GetBytes(crypto.Base64Encode($"{pointClient}: {str}"));
                        s.Send(sendBytes);
                    }
                }
                catch (SocketException ex)
                {
                    clientSockets.Remove(pointClient);
                    mainForm.UserListRemoveItem(pointClient);

                    PrintlnAndSave($"Klient {socketClient.RemoteEndPoint} przerwał
połączenie: {ex.Message}");
                    socketClient.Close();
                    break;
                }
                catch (Exception ex)
                {
                    PrintlnAndSave($"Błąd: {ex.Message}");
                }
```

```
            }
        }
```

7. WYSYŁANIE WIADOMOŚCI

Analogicznie jak w przypadku odbierania wiadomości odbywa się tym razem kompresja na zabezpieczony ciąg znaków za pomocą BASE64. Następnie taka wiadomość wysyłana jest do wszystkich podłączonych klientów.

```
        static void SendMessage(object sender, EventArgs e)
        {
            string message = mainForm.GetMessageString();
            if (message == "")
            {
                return;
            }
            byte[] sendBytes = Encoding.UTF8.GetBytes(crypto.Base64Encode($"Serwer:
{message}"));
            foreach (Socket s in clientSockets.Values)
            {
                s.Send(sendBytes);
            }
            PrintlnAndSave($"Server: " + message);
            mainForm.ClearMessageText();
        }
```

8. ZAPISYWANIE LOGÓW I WYŚWIETLANIE WIADOMOSCI

Serwer zapisuje wszystkie powiadomienia, wiadomości, kody błędów w pliku log.txt (w tym przypadku jego domyślna lokalizacja to: *C:\Users\Dell\source\repos\PROJEKT_CHAT1_SERVER\PROJEKT_CHAT1_SERVER\bin\Debug*, ale można ją zmienić).

Tekst początkowo przesyłany jest do metody PrintlnAndSave. Metoda ta:

- Wyświetla otrzymane ciągi znaków w polu komunikatów za pomocą metody Form1.Println,
- Zapisuje komunikaty do pliku z wykorzystaniem metody WriteLine z klasy SaveLogs (można ją rozwijać).

```
       static void PrintlnAndSave(string str)
              {
                  mainForm.Println(str);
                  saveLogs.WriteLine(str);
              }
```

Metoda Println z Form1. Dostęp z wykorzystaniem delegatów.

```
        public void Println(string str)
        {
            if (this.textLog.InvokeRequired)
            {
                VoidString println = Println;
                this.textLog.Invoke(println, str);
            }
            else
            {
                this.textLog.AppendText(str + Environment.NewLine);
            }
        }
```

Klasa SaveLogs, wraz z użytą metodą do zapisywania do plików:

```
class SaveLogs
    {
        public void WriteLine(String str)
        {
            StringBuilder simpleLog = new StringBuilder();
            simpleLog.Append(DateTime.Now.ToString("yyyy.MM.dd HH:mm:ss"));
            simpleLog.Append((char)9);
            simpleLog.Append(str);

            StreamWriter writer = new StreamWriter("logs.txt", true);
            writer.WriteLine(simpleLog.ToString());
            writer.Close();
        }
    }
```

Jak można zauważyć klasa porządkuje otrzymane wiadomości, dodając czas otrzymania oraz tabulator (tak by komunikaty były czytelniejsze).

Przykładowy fragment logów wygląda następująco:

```
2018.06.20 21:12:43	----------
2018.06.20 21:12:43	Włączono serwer pod adresem: 127.0.0.1:5000
2018.06.20 21:12:48	Server: jolki
2018.06.20 21:12:59	127.0.0.1:50878 klient połączony
2018.06.20 21:13:02	127.0.0.1:50878: : Anonim >> polki
2018.06.20 21:13:15	127.0.0.1:50880 klient połączony
2018.06.20 21:13:21	127.0.0.1:50880: : Antek :D >> polki 2
```

Linia z “----------” jest dodawana z pominięciem metody PrintlnAndSave i jest dodawana przy każdym uruchomieniu serwera, tak by każda sesja była jeszcze bardziej czytelna.