

# PROJEKT\_CHAT1\_Client – DOKUMENTACJA

---

Autor: Krzysztof Rudzki

Język: C#

Połączenie: TCP (sockets, threads)

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

Repozytorium: [https://github.com/krzysieker/PROJEKT\\_CHAT1\\_CLIENT](https://github.com/krzysieker/PROJEKT_CHAT1_CLIENT)

## Spis treści

1. UŻYTY PROTOKÓŁ .....	2
2. OKNO APLIKACJI .....	2
3. POŁĄCZENIE .....	5
4. SZYFROWANIE WIADOMOŚCI .....	5
5. ODBIERANIE WIADOMOŚCI .....	6
6. WYSYŁANIE WIADOMOŚCI .....	7

# 1. UŻYTY PROTOKÓŁ

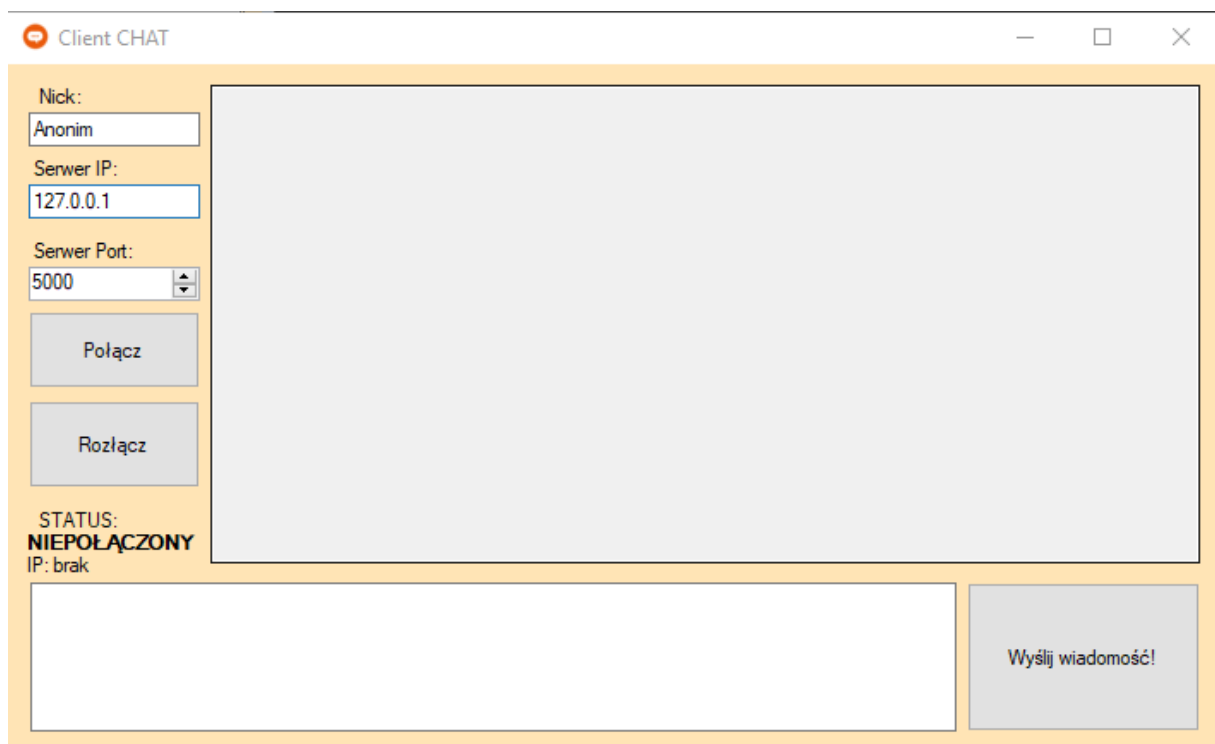
Do połączenia użyto protokołu TCP na socketach, z wykorzystaniem wątków.

```
addressIP = IPAddress.Parse(mainForm.GetIPString());  
ipEndPoint = new IPEndPoint(addressIP, mainForm.GetPortNumber());  
  
socketServer = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
```

# 2. OKNO APLIKACJI

Po uruchomieniu klienta 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 chat roomy.

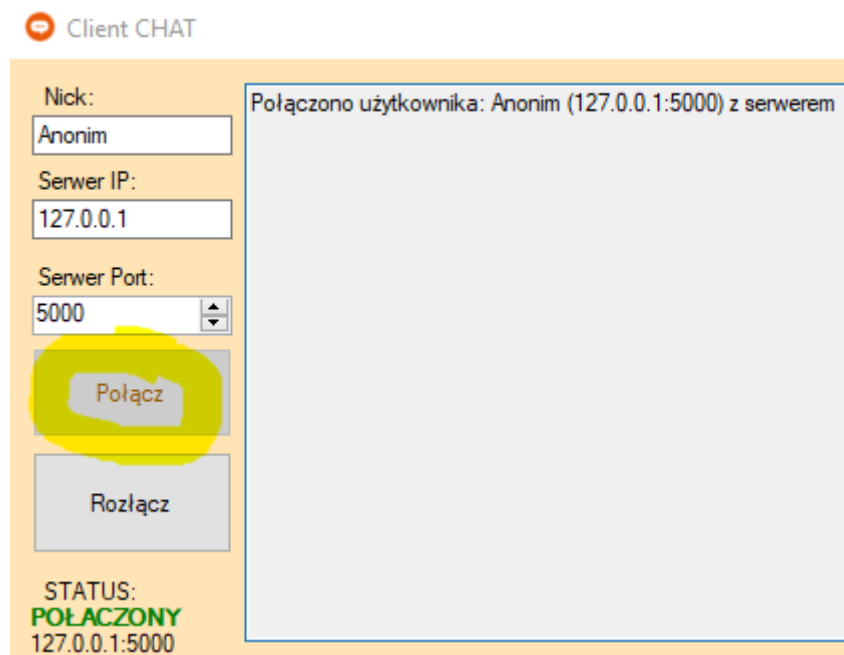
Okno po uruchomieniu wygląda następująco:



- **TextBox: Nick:** Wprowadzenie nicku jakim będziemy posługiwać się podczas komunikacji. Domyślnie jest to „Anonim”. Taki użytkownik jednak anonimowy nie jest, gdyż oprócz nicku klient przedstawia się także adresem IP oraz portem.

- **TextBox: Server IP:** służy do wprowadzenia adresu IP naszego serwera. Domyślnie ustawiono wartość 127.0.0.1, gdyż serwer uruchomiony był na tym samym urządzeniu co klienci. W przypadku lokalnej sieci IP serwera będzie inne.
- **TextBox: Port:** analogicznie do adres IP. Domyślnie zarówno na serwerze jak i kliencie ustawiono jego wartość na 5000,
- **Button: Połącz:** tak jak nazwa wskazuje służy do połączenia z serwerem.

\*Po połączeniu z serwerem button : Połącz zostaje zablokowany.



Kod programu odpowiadający za tą funkcję.

```
public void SetConnectEnabled(bool enabled)
{
    if (this.buttonConnect.InvokeRequired)
    {
        VoidBool sce = SetConnectEnabled;
        this.textLog.Invoke(sce, enabled);
    }
    else
    {
        this.buttonConnect.Enabled = enabled;
    }
}
```

\*Ponieważ funkcje głównego formularza wywoływane są w głównej klasie Program.cs, użyto delegatów.

- **Button: Rozłącz:** tak jak nazwa wskazuje służy do rozłączenia z serwerem. Analogicznie jak w przypadku „Połącz”

```
public void SetDisconnectEnabled(bool enabled)
{
    if (this.buttonDisconnect.InvokeRequired)
    {
        VoidBool sde = SetDisconnectEnabled;
        this.textLog.Invoke(sde, enabled);
    }
    else
    {
        this.buttonDisconnect.Enabled = enabled;
    }
}
```

- **Labels: Status:** Wyświetlają aktualny status klienta oraz adres IP.

Dla przykładu:

- Klient połączony:

```
STATUS:
POŁĄCZONY
127.0.0.1:5000
```

- Klient rozłączony:

```
STATUS:
NIEPOŁĄCZONY
BRAK
```

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

```
Połączono użytkownika: Anonim (127.0.0.1:5000) z serwerem
Rozłączono z serwerem
Serwer został rozłączony: Nawiązane połączenie zostało przerwane przez oprogramowanie zainstalowane w komputerze goście
Połączono użytkownika: Anonim (127.0.0.1:5000) z serwerem
127.0.0.1:51485: : Anonim >> Taki oto komunikat!
```

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

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

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

### 3. POŁĄCZENIE

Tak jak wspomniano przy omawianiu protokołu użyto TCP na socketach, z użyciem wątków.

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

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

    try
    {
        // Nawiązanie połączenia
        socketClient.Connect(iPEndPoint);
        mainForm.SetStatusLabel(true, iPEndPoint.ToString());
        mainForm.SetSendEnabled(true);
        mainForm.SetConnectEnabled(false);
        mainForm.SetDisconnectEnabled(true);
        mainForm.Println($"Połączono użytkownika: {nick} ({iPEndPoint}) z
serverem");

        // Ciągłe odbieranie wiadomości wysyłanych z serwera
        Thread thread = new Thread(Receive);
        thread.IsBackground = true;
        thread.Start(socketClient);
    }
    catch (Exception ex)
    {
        mainForm.Println("Error: " + 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);
mainForm.Println(str);

```

## 5. ODBIERANIE WIADOMOŚCI

Poniżej znajduje się listing odpowiedzialny za odbieranie wiadomości

- 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.
- W przypadku błędów klient jest rozłączany (dla bezpieczeństwa).

```

static void Receive(object obj)
{
    Socket socketSend = obj as Socket;
    while (true)
    {
        try
        {
            // Otrzymuj wiadomości wysłane
            byte[] buffor = new byte[1024];
            int len = socketSend.Receive(buffor);
            if (len == 0)
            {
                break;
            }
            string encodedStr = Encoding.UTF8.GetString(buffor, 0, len);
            string str = crypto.Base64Decode(encodedStr);
            mainForm.Println(str);
        }
        catch (Exception ex)
        {
            mainForm.SetStatusLabel(false);
            mainForm.SetSendEnabled(false);
            mainForm.SetConnectEnabled(true);
            mainForm.Println($"Serwer został rozłączony: {ex.Message}");
            break;
        }
    }
}

```

## 6. 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 = ": " + nick + " >> " + mainForm.GetMessageString();
    if (message == "")
    {
        return;
    }
    byte[] sendBytes = Encoding.UTF8.GetBytes(crypto.Base64Encode(message));
    socketClient.Send(sendBytes);
    mainForm.ClearMessageText();
}
```