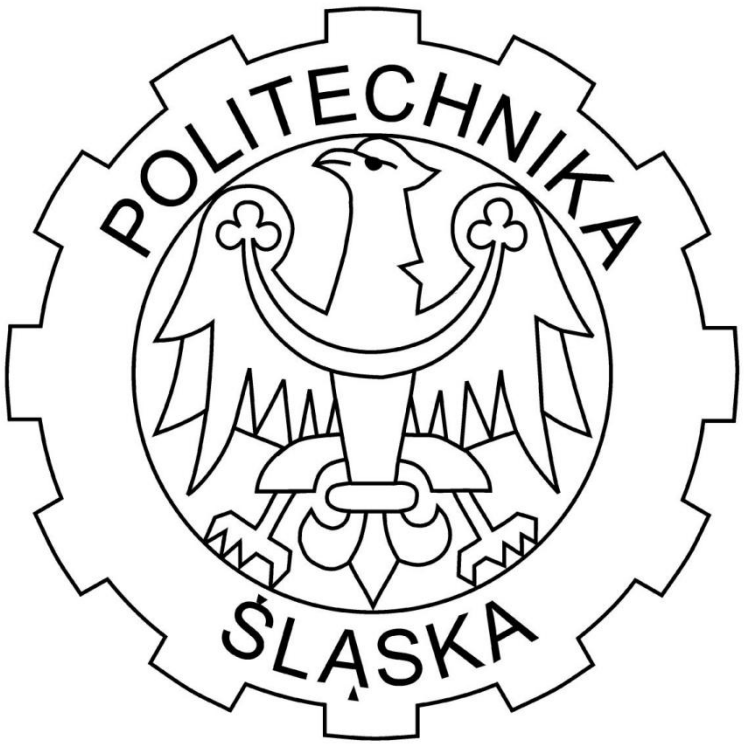
PROGRAMOWANIE W ŚRODOWISKU SIECIOWYM

DOKUMENTACJA PROJEKTU

Prosty komunikator tekstowy z interfejsem graficznym



Politechnika Śląska w Gliwicach
Wydział Elektryczny

Wykonanie:

Adam Celak, Bartłomiej Stokowy

Semestr: **VI**

Kierunek: **Informatyka**

Grupa: **E**

Data wykonania projektu: **Wrzesień 2020r.**

Spis treści

Wstęp do projektu **3**

Cel projektu 3

Założenia projektu 3

Wykorzystane technologie 3

Opis projektu **4**

Uruchomianie 4

Opis serwera 5

Opis klienta 6

Komunikacja 7

Oprogramowanie **8**

Opis kodu 8

Protokół komunikacyjny 10

Niniejszy dokument stanowi dokumentację projektu „Prosty komunikator tekstowy z interfejsem graficznym" stworzonego przez studentów Politechniki Śląskiej na potrzeby przedmiotu Programowanie W Środowisku Sieciowym.

1. WSTĘP DO PROJEKTU

a) Cel projektu

Celem projektu jest stworzenie prostego komunikatora tekstowego umożliwiającego wymianę informacji pomiędzy dwoma użytkownikami za pośrednictwem serwera

b) Założenia projektu

Projektowana aplikacja będzie składać się z dwóch części: klienta oraz serwera.

- Serwer nie będzie wyposażony w interfejs użytkowy będzie to jedynie program umożliwiający komunikację pomiędzy dwoma klientami. Wgląd w funkcjonowanie serwera będzie możliwy tylko za pomocą konsoli
- Kliencka część aplikacji zostanie wyposażona w prosty interfejs graficzny obsługiwany za pomocą klawiatury i myszy. Użytkownicy będą mieli możliwość nadania sobie nazwy i wysyłania wiadomości.

c) Wykorzystane technologie

Do stworzenia projektu wykorzystany zostanie język C++ wraz z bibliotekami Qt.

Środowisko w jakim zostanie napisany program to Visual Studio 2017 z dodatkami przeznaczonymi do projektowania z Qt.

Aplikacja będzie działała co najmniej na systemach Microsoft Windows 2007 lub nowszych.

2. OPIS URZĄDZEŃ

Niniejszy rozdział zawiera informacje dotyczące instalacji oraz użytkowania programu oraz wyjaśnienie głównych funkcji programu.

a) Uruchomienie

Instalacja programu polega na rozpakowaniu skompensowanego folderu .rar w dowolne miejsce na komputerze użytkownika. Następnie w celu umożliwienia wymiany informacji należy w pierwszej kolejności uruchomić serwer (Katalog: Server > Server.exe) a następnie można uruchomić klienta *lub dwóch* (Katalog: Client > Client.exe).

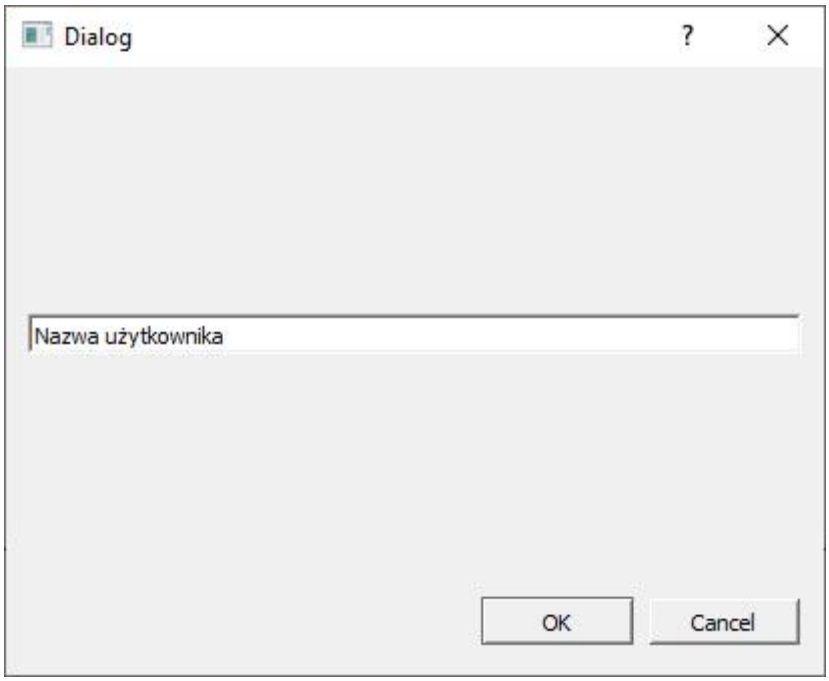
b) Opis serwera

Po uruchomieniu aplikacji serwera otwiera się puste okno konsolowe (Rys.1), jest to oczekiwane działanie programu. Okna tego nie należy zamykać gdyż uniemożliwi to komunikację pomiędzy klientami (można je zminimalizować).

Rys. 1. Serwer

c) Opis klienta

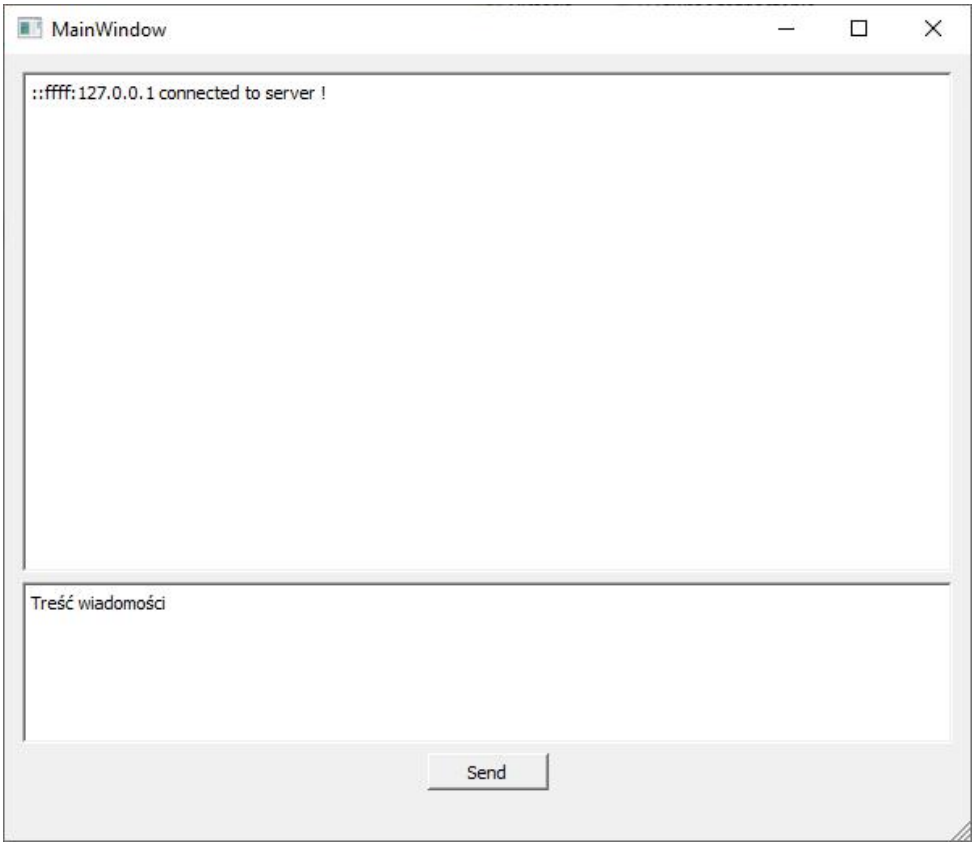
Po uruchomieniu aplikacji klienta jako pierwsze pojawia się okno w którym należy wprowadzić nazwę użytkownika (Rys.2). Po w wprowadzeniu nazwy należy nacisnąć przycisk „OK”. Nie wprowadzenie nazwy lub kliknięcie przycisku „Cancel” przerwie działanie aplikacji.



Rys. 2. Okienko wprowadzania nazwy

W kolejnym oknie widnieją dwa pola tekstowe (Rys.3). Górne, większe pole wyświetla historię czatu, natomiast dolne przeznaczone jest na wpisanie treści wiadomości. Aby wysłać wiadomość należy kliknąć na przycisk „Send” znajdujący się na dole okna aplikacji *(przycisk Enter wciśnięty na klawiaturze nie powoduje wysłania wiadomości).*

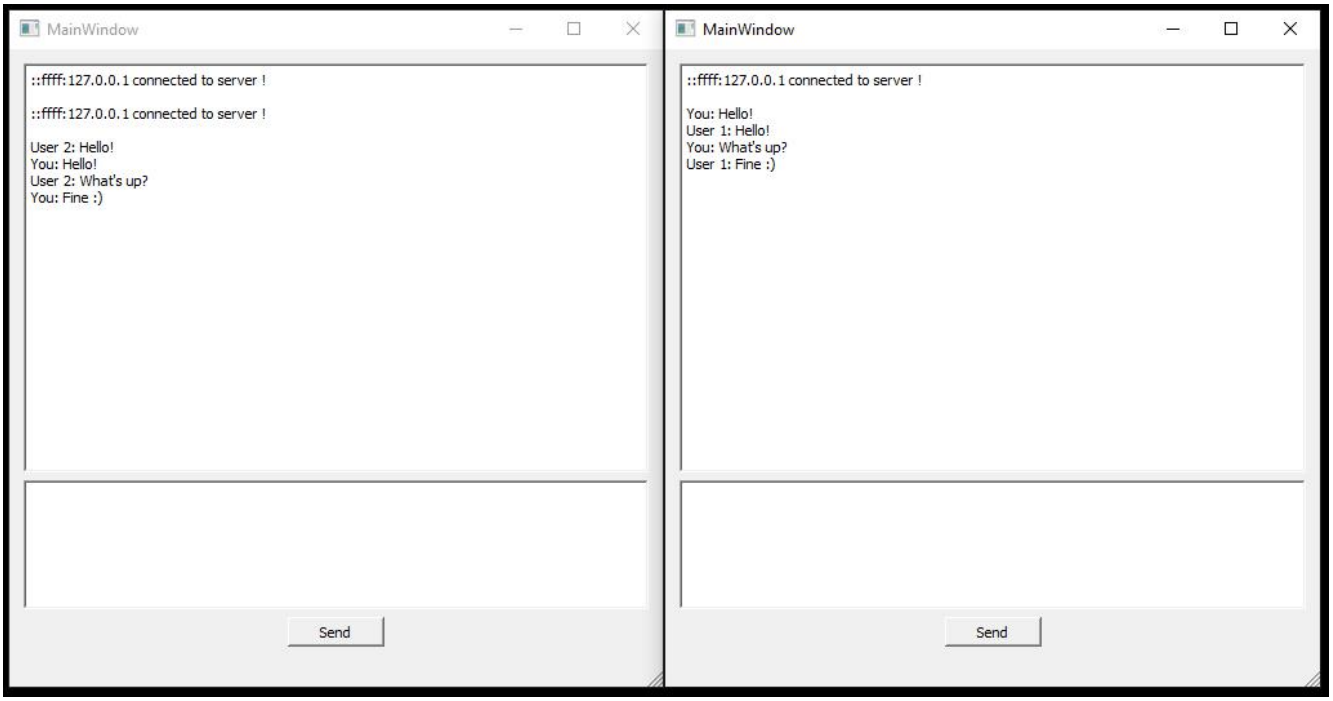
Oba okna aplikacji są w pełni skalowalne.



Rys. 3. Okienko czaru

d) Komunikacja

Komunikacja odbywa się między użytkownikami za pośrednictwem serwera. Poniżej przedstawiono przykładową komunikację pomiędzy użytkownikami (Rys. 4) oraz jej reprezentacje na serwerze (Rys. 5).



Rys. 4. Komunikacja między 2 klientami

Rys. 5. Komunikacja (widok serwera)

3. OPROGRAMOWANIE

a) Opis kodu

Kod napisanej przez nas aplikacji został wyposażony w komentarze informujące o przeznaczeniu i/lub działaniu konkretnych jego fragmentów, dla tego też jako opis kodu zamieszczono listening kilku fragmentów programu. Przedstawiono go na Rysunkach 6-8.

```
#include <QDebug>
#include <QtNetwork/QHostAddress>
#include <QtCore/QTextCodec>
#include "Client.h"

Client::Client() //połączenie z hostem
{
    Socket.connectToHost(QHostAddress("127.0.0.1"), 4242);
    connect(&Socket, SIGNAL(readyRead()), this, SLOT(onReadyRead()));
}

void Client::onReadyRead()  //funkcja potrzebna do przechwytywania wiadomości wysyłanych przez użytkownika
{
    QByteArray data = Socket.readAll();
    QString dataStr = QString(data);
    emit PassDataToMainWindow(dataStr);
}

void Client::SendMessage(QString message) //funkcja odpowiedzialna za wysyłanie wiadomości do serwera
{
    if (Socket.isWritable())
    {
        QString messageToSend = Name + ": " + message;
        Socket.write(messageToSend.toStdString().c_str());
    }
}

void Client::SetUserName(QString name) //ustawienie nazwy użytkownika
{
    Name = name;
}

Client::~Client()
{
}
```

Rys. 6. Listening 1

```cpp
#include "MainWindow.h"

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
{
    ui.setupUi(this);
    connect(ui.sendButton, SIGNAL(clicked()), this, SLOT(OnSendButtonClick()));
    connect(&client, SIGNAL(PassDataToMainWindow(QString)), this, SLOT(GetData(QString)));
    connect(&nameAccepterDialog, SIGNAL(SendExit()), this, SLOT(CloseApplication()));
    connect(&nameAccepterDialog, SIGNAL(SendName(QString)), this, SLOT(GetName(QString)));
}

void  MainWindow::ShowNameAccepter() //pokazywanie nazw użytkowników
{
    this->setEnabled(false);
    nameAccepterDialog.show();
}

void MainWindow::GetData(QString data) //odbieranie danych
{
    ui.incomingEdit->append(QString(data));
}

void MainWindow::CloseApplication() //zamykanie aplikacji
{
    nameAccepterDialog.close();
    this->setEnabled(true);
    QMainWindow::close();
    QApplication::quit();
}

void MainWindow::GetName(QString name) //ustawianie nazwy
{
    client.SetUserName(name);
    nameAccepterDialog.close();
    this->setEnabled(true);
}

void MainWindow::OnSendButtonClick() //gdy zostanie wciśnięty przycisk wysyła się wiadomość
{
    QString Message = ui.outcomingEdit->toPlainText();
    ui.outcomingEdit->clear();
    ui.incomingEdit->append(QString("You: " + Message));
    client.SendMessage(Message);
}
```

Rys. 7. Listening 2

```cpp
#include <QDebug>
#include <QtNetwork/QHostAddress>
#include <QtNetwork/QAbstractSocket>
#include "Server.h"

Server::Server()
{
    TcpServer.listen(QHostAddress::Any, 4242);
    connect(&TcpServer, SIGNAL(newConnection()), this, SLOT(OnNewConnection()));
}

void Server::OnNewConnection() //informacja gdy połączy się nowy użytkownik
{
    QTcpSocket *clientSocket = TcpServer.nextPendingConnection();

    connect(clientSocket, SIGNAL(readyRead()), this, SLOT(OnReadyRead()));
    connect(clientSocket, SIGNAL(stateChanged(QAbstractSocket::SocketState)), this, SLOT(OnSocketStateChanged(QAbstractSocket::SocketState)));

    Sockets.push_back(clientSocket);
    for (QTcpSocket* socket : Sockets) {
        socket->write(QByteArray::fromStdString(clientSocket->peerAddress().toString().toStdString() + " connected to server !\n"));
    }
}

void Server::OnSocketStateChanged(QAbstractSocket::SocketState socketState) //użytkownik jest usuwany gdy się rozłączy
{
    if (socketState == QAbstractSocket::UnconnectedState)
    {
        QTcpSocket* sender = static_cast<QTcpSocket*>(QObject::sender());
        Sockets.removeOne(sender);
    }
}

void Server::OnReadyRead() //serwer przechowuje wiadomości przesyłane przez użytkownika
{
    QTcpSocket* sender = static_cast<QTcpSocket*>(QObject::sender());
    QByteArray datas = sender->readAll();
    qDebug() << datas;
    for (QTcpSocket* socket : Sockets) {
        if (socket != sender)
            socket->write(datas);
    }
}

Server::~Server()
{
}
```

Rys. 8. Listening 3

b) Opis protokołu komunikacyjnego

Projekt został opary o protokół komunikacyjny TCP. Jest to protokół komunikacyjny stosowany do przesyłania danych między komputerami w sieciach. TCP działa w trybie klient-serwer. Serwer oczekuje na połączenie na konkretnym porcie. Klient natomiast inicjuje połączenie do serwera. TCP gwarantuje dostarczenie wszystkich pakietów w całości(bez duplikatów), wraz z zachowaniem kolejności. TCP operuje w warstwie transportowej modelu OSI.