Bielsko-Biała, 30.04.2019 r.



Asynchroniczny komunikator client – server z komunikacją po TCP/IP

Autor: Marek Pisarek

Spis treści

[Opis i zasada działania projektu 3](#_Toc7523936)

[Szczegóły i wymagania dotyczące projektu 3](#_Toc7523937)

[Uruchomienie aplikacji 3](#_Toc7523938)

[Opis interfejsu sieciowego 3](#_Toc7523939)

[Budowa aplikacji Client / Server 5](#_Toc7523940)

[Szczegółowe omówienie budowy aplikacji client 5](#_Toc7523941)

[Szczegółowe omówienie budowy aplikacji server 11](#_Toc7523942)

[Bibliografia 16](#_Toc7523943)

# Opis i zasada działania projektu

## Szczegóły i wymagania dotyczące projektu

Projekt składa się z dwóch aplikacji: klienta i serwera które realizują funkcję komunikatora (przesył wiadomości) w sposób asynchroniczny – zarówno klient jak i serwer nie wymagają współistnienia komunikujących się procesów w tym samym czasie. Komunikaty na nic nie czekają wysyłane są do bufora i są z niego pobierane. Komunikaty zarówno przesyłane jak i odbierane z bufora są zgodne z kodowaniem UTF-8 (uwzgledniają znaki diakrytyczne). Obie aplikację posiadają GUI utworzone w Windows Presentation Foundation.

Zadania które realizuje serwer:

* Tworzy socket służący do komunikacji: IP + Port,
* Wymienia informację z klientem,
* Umożliwia zapis zawartości konsoli (wiadomości) do pliku tekstowego

Zadania które realizuje klient:

* Łączy się z otwartym socketem: IP + Port,
* Umożliwia przekazanie wiadomości do serwera

## Uruchomienie aplikacji

Wraz z dokumentacją została dostarczona solucja zawierająca projekt aplikacji oraz serwera. W folderze /bin/ znajdują się pliki wykonywalne .exe które zawierają skompilowaną solucje. Wymagania środowiska uruchomieniowego to platforma .Net wersja 3.0 – 4.8 która zawiera przestrzenie nazw wymagane do uruchomienia projektu:

System.Net / System.Net.Sockets / System.Component.Model / System.Text / System.Threading

Oraz silnik graficzny który wyrenderuje stworzony interfejs.

## Opis interfejsu sieciowego

TCP określa, w jaki sposób aplikacje mogą tworzyć kanały komunikacji w sieci. Zarządza również sposobem, w jaki wiadomość jest składana w mniejsze pakiety, zanim zostaną one przesłane przez Internet i złożone w odpowiedniej kolejności pod adresem docelowym.

IP określa sposób adresowania i trasowania każdego pakietu, aby upewnić się, że dotrze on do właściwego miejsca docelowego. Każdy komputer bramy w sieci sprawdza ten adres IP, aby określić, gdzie przesłać wiadomość dalej. Natomiast Port jest identyfikatorem aplikacji lub serwisu uruchamianego na urządzeniu.

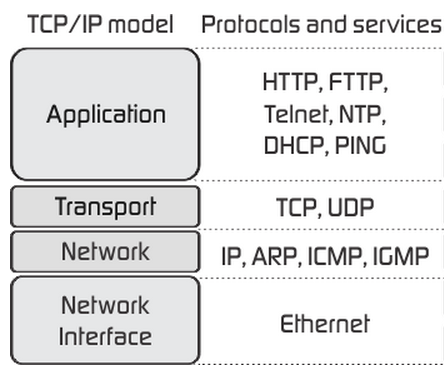


Diagram 1 Warstwy modelu TCP/IP

W połączeniu TCP – Socket jest punktem końcowym połączenia na urządzeniu (end point). Składa się on z adresu IP + przypisanego Portu. Zaprogramowanie socketów polega na sposobie połączenia dwóch punktów końcowych w sieci tak aby się ze sobą komunikowały. Jeden socket (server) nasłuchuje na danym porcie i adresie ip natomiast drugi się z nim łączy tak by ustanowic połączenie.

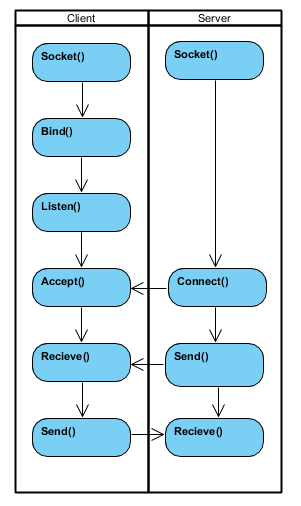


Diagram 2 TCP Socket flow diagram

# Budowa aplikacji Client / Server

Projekt został wykonany w języku programowania C# z wykorzystaniem frameworku WPF który również korzysta z Extensible Application Markup Language (XAML) do tworzenia widoku w aplikacji. Projekty zostałe oparte na wzorcu architektonicznym MVVM (Model-View-ViewModel) oraz interfejsie InotifyPropertyChanged który jest używany do DataBindingu czyli łączenia wizualnych kontrolek z właściwościami obiektów oraz ich odświeżania przy pomocy zdarzenia PropertyChanged.

W obydwu projektach zostały wykorzystane przestrzenie nazw:

System.ComponentModel – zawierająca interfejs INotifyPropertyChanged.

System.IO – zawierająca klase File która dostarcza metody odpowiedzialne za operacje na plikach

System.Net – zawierająca klasę IPEndPoint która przedstawia punkt końcowy sieci jako ip adress i numer portu oraz klasę IPAddress zawierającą metode Parse która zmienia typ string na instancje IPAddress.

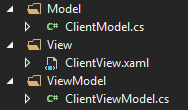
System.Net.Sockets – zawiera klasę Socket która pozwala na implementacje interfejsu Socketu Berkeley’a.

System.Text – zawiera klasę odpowiedzialną za kodowanie tekstu w standardzie UTF-8 oraz klasę StringBuilder służącą do łączenia łańcuchów znaków z bufora.

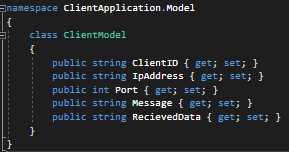
System.Threading – zawiera klasę odpowiadającą za obsługe i tworzenie wątków.

## Szczegółowe omówienie budowy aplikacji client

Struktura aplikacji klient:



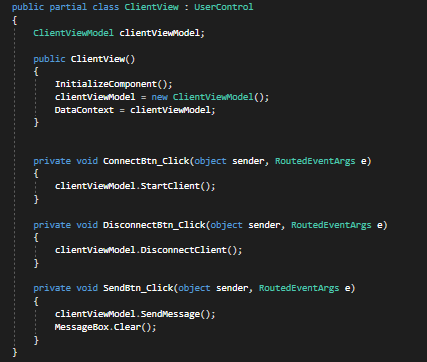
Zawartość klasy Model:



Model aplikacji jest reprezentowany przez klasę zawierającą:

* ClientID (string) – generowany GUID (6 znakowy) reprezentujący klienta,
* IpAddress (string) – adres IP który wprowadza użytkownik wymagany do utworzenia socketu,
* Port (int) – numer portu który wprowadza użytkownik wymagany do utworzenia socketu,
* Message (string) – przechowywanie wiadomości użytkownika jako ciągu znaków,
* RecievedData (string) – przechowywanie wiadomości otrzymywanych w konsoli jako ciąg znaków.

Zawartość widoku (ClientView) utworzonego jako UserControl



Codebehind zawiera przypisanie zawartości UserControl do klasy ViewModel oraz utworzenie jej instancji.

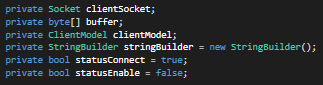
Zdarzenie Click na przycisku Connect wywołuje metode StartClient() z klasy ViewModel.

Zdarzenie Click na przycisku Disconnect wywołuje metode DisconnectClient() z klasy ViewModel.

Zdarzenie Click na przycisku Send Message wywołuje metode SendMessage() z klasy ViewModel oraz usuwa aktualną zawartość TextBoxu przy pomocy metody Clear().

Zawartość klasy ClientViewModel

Właściwości klasy:



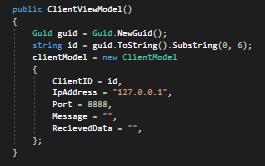
* clientSocket (Socket) – odpowiada za wszystkie operacje związane z socketem który jest tworzony na aplikacji klienta.
* buffer (byte[]) – zawierający wiadomość przesyłaną do serwera w formacie tablicy bajtów
* clientModel (ClientModel) – obsługujący wszystkie właściwości klasy ClientModel
* stringBuilder (StringBuilder) – służący do połączenia otrzymywanego łańcucha znaków
* statusConnect oraz statusEnable (bool) – statusy obsługujące włączenie / wyłączenie zawartości w widoku.

Akcesory get / set do pól prywatnych

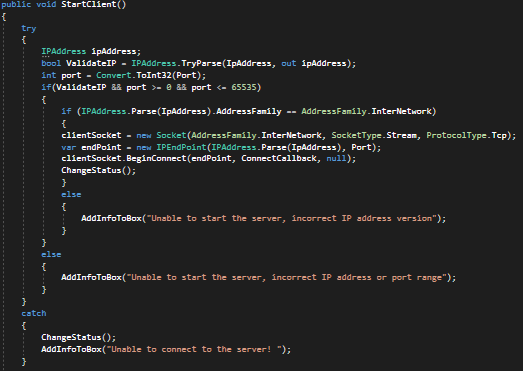


Z nich pobieramy i ustawiamy zawartość pól prywatnych występujących w klasie ViewModel. Podczas ustawienia zawartości jest również wywoływana metoda OnPropertyChanged należąca do interfejsu INotifyPropertyChanged która informuje aplikacje o zmianie wartości w modelu i odświeżeniu UI.

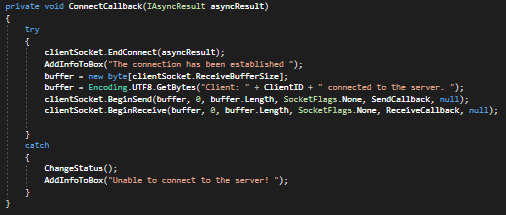
Konstruktor klasy ClientViewModel



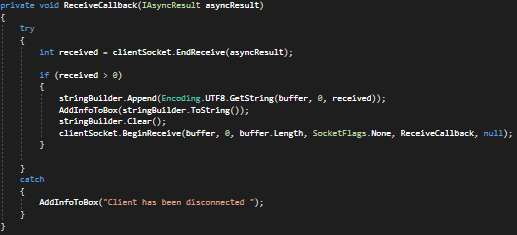
Zostaje utworzona instancja klasy ClientModel z wygenerowanym losowym (6-cio znakowym) ClientID oraz przypisanymi domyślnymi wartościami do klasy.



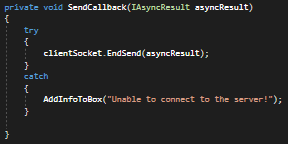
Metoda StartClient odpowiada za utworzenie socketu klienta z parametrami (IPv4, Two-Way connection wraz z protokołem transmisji TCP). Tworzony jest również endPoint na podstawie wprowadzonego adresu IP oraz Portu i zostaje on przypisany do Socketu klienta. Metodą BeginConnect jest wysyłany asynchroniczny request do socketu utworzonego na serwerze. Po udanym połączeniu zmieniany jest status kontrolek w widoku (IsEnabled). W metodzie również sprawdzana jest poprawność wprowadzanych danych (czy adres ip jest w wersji V4 oraz nie zawiera nieprawidłowych znaków oraz czy numer portu mieści się w dostępnym zakresie).



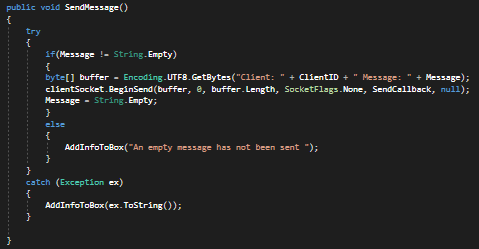
Metoda ConnectCallback kończy asynchroniczny connection request po uzyskaniu połączenia. Zostaje dodana informacja do konsoli, że zostało ustanowione połączenie. Zostaje utworzony bufor i tworzona jest informacja zwrotna do serwera o udanym połączeniu clienta o danym ID która jest wysyłana asynchroniczną metodą (BeginSend) do socketu. Następnie rozpoczyna asynchroniczny odbiór danych z połączonego socketu (BeginReceive)



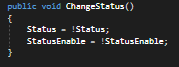
Metoda RecieveCallback kończy asynchroniczny odczyt z socketu, odkodowywuje zawartość z bufora do formatu UTF8, łańcuch znaków zostaje połączony przy użyciu stringBuildera. Socket klienta rozpoczyna asynchroniczny odbiór danych z socketu (BeginRecieve). Zostaje dodana zawartość stringBuildera do konsoli i zostaje on wyczyszczony (Clear)



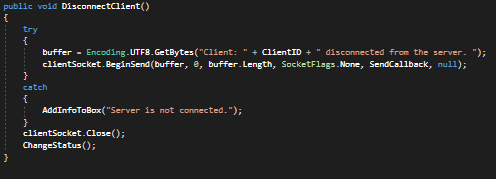
Metoda callback która kończy asynchroniczny przesył.



Metoda SendMessage odpowiedzialna za pobranie znaków z TextBoxu (jeżeli pole na wiadomość nie jest puste) dodanie wiadomości do bufora i wysłanie jej metodą BeginSend. Następnie string przechowujący wiadomość (Message) zostaje czyszczony.



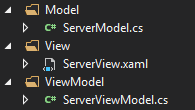
Metoda ChangeStatus odpowiedzialna za zmiane statusów zbindowanych w widoku do IsEnabled tak by kontrolować wyłączenie odpowiednich elementów UI.



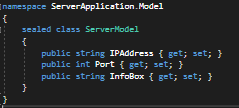
Metoda DisconnectClient która wywoływana jest z poziomu UI zamyka otwarty socket i jeżeli jest on połączony z serwerem przesyła informacje o rozłączeniu danego klienta.

## Szczegółowe omówienie budowy aplikacji server

Struktura aplikacji serwer:



Zawartość klasy Model:

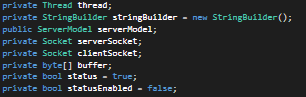


Model aplikacji jest reprezentowany przez klasę zawierającą:

* IPAddress (string) – wprowadzany przez użytkownika, wymagany do utworzenia socketu,
* Port (int) – wprowadzany przez użytkownika, wymagany do utworzenia socketu,
* InfoBox (string) – ciąg znaków przechowujący otrzymywane wiadomości.

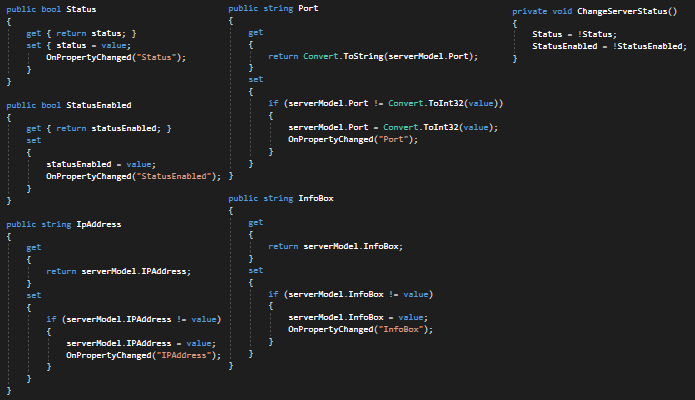
Zawartość klasy ViewModel:

Właściwości klasy ViewModel



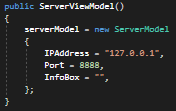
* thread (Thread) – odpowiada za utworzenie wątku obsługującego metode sprawdzającą czy połączony client nie zerwał połączenia
* serverSocket (Socket) – odpowiada za wszystkie operacje związane z socketem który jest tworzony w aplikacji serwera.
* clientSocket (Socket) – odpowiada za ustanowienie połączenia pomiędzy socketami.
* buffer (byte[]) – zawierający wiadomość przesyłaną/odbieraną w formacie tablicy bajtów
* serverModel (ServerModel) – obsługujący wszystkie właściwości klasy ServerModel
* stringBuilder (StringBuilder) – służący do połączenia otrzymywanego łańcucha znaków
* status oraz statusEnabled (bool) – statusy obsługujące włączenie / wyłączenie zawartości w widoku.

Akcesory get / set pól prywatnych w klasie ViewModel:

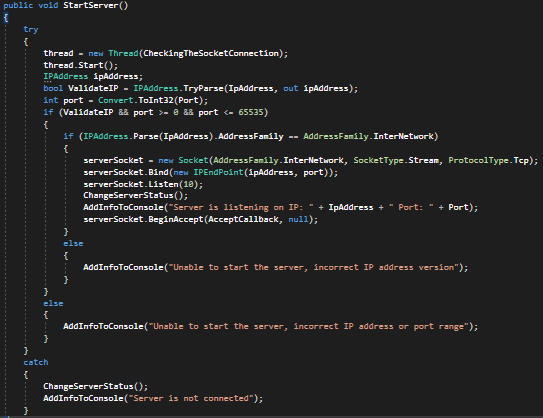


Z nich pobieramy i ustawiamy zawartość pól prywatnych występujących w klasie ViewModel. Podczas ustawienia zawartości jest również wywoływana metoda OnPropertyChanged należąca do interfejsu INotifyPropertyChanged która informuje aplikacje o zmianie zawartości i odświeżeniu UI.

Konstruktor klasy ServerViewModel

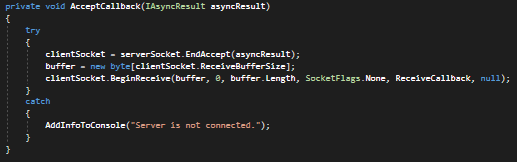


Tworzący instancję klasy ServerModel, następuje przypisanie domyślnych wartości do klasy.



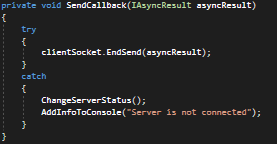
Metoda StartServer odpowiada za utworzenie socketu serwera z parametrami (IPv4, Two-Way connection wraz z protokołem transmisji TCP). Tworzony jest również endPoint na podstawie wprowadzonego adresu IP oraz Portu i zostaje on przypisany do Socketu serwera. Socket serwera rozpoczyna nasłuchiwanie, następuje zmiana statusów (przełączenie kontrolek). Zostaje umieszczona informacja w konsoli na którym porcie i adresie IP nasłuchuje serwer. Socket serwera jest umieszczony w stanie akceptacji przychodzących połączeń BeginAccept. Również w tej metodzie tworzony jest wątek który rozpoczyna sprawdzanie czy klient po podłączeniu się z socketem nie przerwał swojego połączenia oraz walidacja wprowadzonych danych (IP wersja V4 w poprawnej formie oraz port w dostępnym przedziale 0-65535)

Metoda AcceptCallback



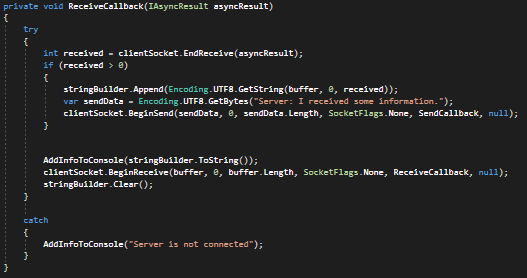
Zostaje wywołana w przypadku połączenia się klienta z serwerem. Socket serwera akceptuje przychodzące połączenie i tworzy socket clienta żeby utrzymać zdalną komunikację. Tworzony jest bufor który jest przekazywany do metody odpowiedzialnej za przyjmowanie danych przychodzących z połączonego socketu.

Metoda SendCallback



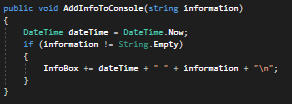
Funkcja zwrotna która kończy asynchroniczny przesył.

Metoda ReceiveCallback



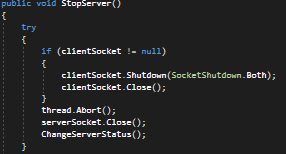
Metoda RecieveCallback wywoływana w przypadku otrzymanai wiadomości, kończy asynchroniczny odczyt z socketu, odkodowywuje zawartość z bufora do formatu UTF8, łańcuch znaków zostaje połączony przy użyciu stringBuildera. Socket serwera ponownie rozpoczyna asynchroniczny odbiór danych z socketu (BeginRecieve). Zostaje dodana zawartość stringBuildera do konsoli i zostaje on wyczyszczony (Clear)

Metoda AddInfoToConsole



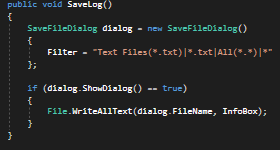
Formatuje wiadomość w sposób następujący: Aktualna data + czasa + otrzymana wiadomość + przeniesienie do nowej linii i dodaje ją do pola przechowującą wiadomości.

Metoda StopServer



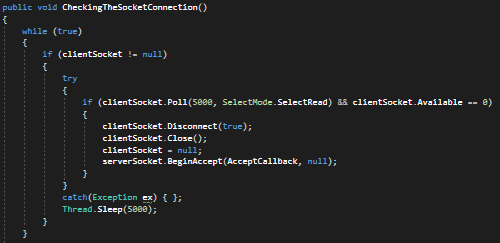
Zamyka utworzone sockety oraz zwalnia zasoby do nich należące.

Metoda SaveLog



Odpowiedzialna za utworzenie pliku tekstowego, przy pomocy klasy SaveFileDialog tworzone jest okno dialogowe w którym użytkownik może zapisać zawartość konsoli serwera w pliku .txt.

Metoda CheckingTheSocketConnection



Polega na sprawdzeniu czy po połączeniu się klienta z serwerem, on nie przerwał swojego działania. Jeżeli klient się rozłączył, zwalniane są jego zasoby a serwer ponownie oczekuje na połączenie.

# Bibliografia

1. <http://www.steves-internet-guide.com/tcpip-ports-sockets/>

2.<https://www.ibm.com/support/knowledgecenter/en/SSB27H_6.2.0/fa2ti_what_is_socket_connection.html>

3. <https://en.wikipedia.org/wiki/Network_socket>

4. <http://ww2.ii.uj.edu.pl/~rytko/SK/SK_2012.pdf>

5. <https://docs.microsoft.com/pl-pl/dotnet/api/system.net.sockets.socket?view=netframework-4.8>

6. <https://docs.microsoft.com/pl-pl/dotnet/csharp/>

7. <https://www.geeksforgeeks.org/tcp-server-client-implementation-in-c/>

8. <https://aticleworld.com/socket-programming-in-c-using-tcpip/>