## **Budowanie projektu serwera z .NET**

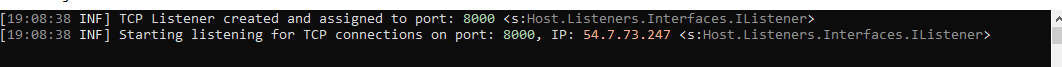
**Repozytorium:** <https://github.com/PablitoCBR/TCP-Chat-Server>

**.NET Core 2.1 Runtime:** <https://dotnet.microsoft.com/download/dotnet-core/thank-you/runtime-2.1.14-windows-x86-installer>

1. Pobierz i zainstaluj .NET Core 2.1 Runtime (może być wymagany restart po instalacji)
2. Sklonuj repozytorium do dowolnej lokalizacji.
3. Używając CMD albo Powershell’a przejdź do lokalizacji ze skopiowanym repozytorium.
4. Będąc w folderze z repozytorium przejdź do folderu **Server.**
5. Będąc w folderze **Server** użyj komendy **„dotnet restore”**
6. Następnie podaj komendę **„dotnet build”**
7. Na koniec do uruchomienia projektu użyj komendy **„dotnet run”**

**! – Po wykonaniu jednorazowym tych kroków wystarczy komenda „dotnet run” do uruchamiania projektu. W przypadku aktualizacji projektu możesz potrzebować ponownie użyć „restore” i „build”.**

! -Po uruchomieniu aplikacja wyświetli informacje do jakiego adresu IP oraz portu został przypisany listener (domyślnie jest to port 800 oraz Twój adres w lokalnej sieci).



! - Od tego momentu Server słucha na podanym porcie przychodzących połączeń TCP.

! - Serwer będzie w sieci lokalnej więc IP będzie przypisywane mu dynamicznie z DHCP co oznacza, że nie uda się nam tego hardcodem dodać w kliencie. Musisz zostawić opcję do konfiguracji klienta na jaki adres IP oraz port (tutaj raczej nie trzeba bo zostawiłbym 8000 ale nice to have) do jakiego ma się odwoływać jako serwera. Może jakiś plik konfiguracyjny w JSON’ie albo XML, który klient przeczyta przy starcie?

## **Serializacja komunikatów do serwera**

Komunikat do serwera dzieli się na dwie podstawowe części **meta dane komunikatu** oraz **komunikat**, w którym zawarte są nagłówki oraz wiadomość.

**META DANE KOMUNIKATU:**

Meta dane określają typ komunikatu oraz długość nagłówków i komunikaty (liczba bajtów w wiadomości).

Długość całej części meta danych to 9 bajtów na co składa się kolejno:

- 1 bajt typu komunikatu (aktualna lista typóww kodzie szestnastkowym wraz z opisem: <https://github.com/PablitoCBR/TCP-Chat-Server/blob/master/Core/Models/Enums/MessageEnums.cs>) wysyłając żądanie do serwera używamy typów **Message user request codes (51 -100)**. Odpowiedzi serwera oznaczające sukces określone są przez **Server successful response codes (1 – 50)** reszta to kody błędów podzielonych na spowodowanych przez klienta przez np.. błędne dane oraz błędy serwerowe. **!!! Kody będą rozwijane oraz nie wszystkie mogą jeszcze działać.**

- 4 bajty określające długość (ilość bajtów) nagłówków [**typ Integer]**

- 4 bajty określające długość (ilość bajtów) wiadomości [**typ Integer]**

**KOMUNIKAT**:

Kontynuuje meta dane i zawiera w sobie **nagłówki oraz wiadomość** o długości określonej w **meta danych.**

**NAGŁÓWKI:**

Zawierają informacje ważne dla komunikatu w postaci słownika (klucz:warość) tak samo jak nagłówki http.

Kolejne nagłówki są od siebie oddzielone znakiem nowej linii **„\n”**, który też musi zostać zakodowany.

Klucze i wartości są oddzielone dwukropkiem **„:”**.

Przykładowe nagłówki(znak nowej linii jest tylko dla przykładu):

**authentication:aswer43YSsdfafasdfdsaf==\n**

**recipient:pawel\n**

Serwer przyjmuje oraz zwraca nagłówki w takim formacie!!!!!

**! – Zdefiniowane rodzaje nagłówków:** <https://github.com/PablitoCBR/TCP-Chat-Server/blob/master/Core/Models/Consts/MessageHeaders.cs>

**WIADOMOŚĆ**:

Wiadomość to zwyczajny string sprowadzony do postaci bajtów….

# **Podstawowe komunikaty do serwera !!**

**REJESTRACJA**

Komunikat z prośbą o rejestracje musi być typu **RegistrationRequest (0x36)**

W nagłówku komunikatu musi zostać zawarty nagłówek **authentication**, którego wartością jest nazwa użytkownika oraz hasło oddzielone dwukropkiem (**username:password**) zapisane w formacie **Base64!!!**

Po zarejestrowaniu użytkownika w bazie serwer zwróci wiadomość z kodem **Registered (0x33)** i zamknie połączenie!

Od tego momentu użytkownik może logować się do serwera używając nazwy usera i hasła z rejstracji.

! – przy błędach z rejestracją zostanie zwrócona wiadomość z kodem **RegistrationFailed (0x66)**, a w wiadomości będzie zapisana informacja o powodzie błędu.

**LOGOWANIE**

Komunikat z prośbą o logowanie musi być typu **AuthenticationRequest (0x37)**

Nagłówek komunikatu musi zawierać nagłówek **authentication**, w formacie takim samym jak przy rejestracji przy użyciu wcześniej zarejestrowanej nazwy usera i hasła!

Po poprawnej autentykacji serwer zwróci komunikat z typu **Authenticated (0x04).**

Serwer nie zamyka połączenia, a utrzymuje je i od tego momentu nasłuchuje na komunikaty od klienta oraz jest gotowy przysyłać komunikaty do klienta!!!

! – przy błędach z rejestracją zostanie zwrócona wiadomość z kodem **Unauthenticated (0x65)**, a w wiadomości będzie zapisana informacja o powodzie błędu.

**WYSYŁANIE WIADOMOŚCI POMIĘDZY KLIENTAMI**

Komunikat z prośbą o logowanie musi być typu **MessageSendRequest (0x33)**

Nagłówek komunikatu musi zawierać nagłówek **recipient** określającego odbiorcę którego wartością jest nazwa użytkownika, który ma być odbiorcą wiadomości.

Wiadomość zawiera wiadomość skierowaną do użytkownika.

Udane dostarczenie wiadomości jest sygnalizowane przez serwer wiadomością zwrotną typu **MessageSent (0x04)**

Wiadomość przychodząca jest typu **EncryptedMessage (0x01)**.

! – wiadomośc zostanie dostarczona pod warunkiem, że odbiorca jest połączony z serwerem. W przeciwnym przypadku zwrotka będzie typu **ClientUnreachable (0xC9).**