KOMUNIKATOR SUPER TIN-TIN

Dokumentacja wstępna

**Zespół nr 3:**  
*Domagała Bartosz  
Kaczor Bartosz  
Łopusiński Paweł  
Stepnowski Marcin*

1. Cel projektu

Celem projektu jest stworzenie prostego tekstowego komunikatora internetowego umożliwiającego prowadzenie wieloosobowych rozmów oraz przesyłanie plików. System będzie posiadał centralny serwer zarządzający zestawianiem połączeń między użytkownikami. System będzie umożliwiał rejestrację użytkownikom, stworzenie własnej listy znajomych oraz zarządzanie nią.

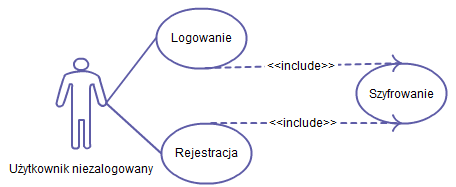
1. Wymagania funkcjonalne
   1. Aplikacja kliencka – aktor: Użytkownik

* Rejestracja – aplikacja umożliwia użytkownikowi założenie konta na serwerze. Użytkownik wybiera login (musi być unikatowy) i hasło a system samodzielnie przypisuje mu unikatowy identyfikator.
* Logowanie – użytkownik podaje login i hasło wcześniej utworzonego konta. Serwer przeprowadza uwierzytelnianie – jeśli zakończone jest powodzeniem, użytkownik uzyskuje dostęp do pełnej funkcjonalności aplikacji.
* Wylogowanie – zakończenie sesji uwierzytelnionej.
* Wyszukiwanie użytkowników – system umożliwia przeszukanie bazy użytkowników serwera (po ID lub loginie).
* Zarządzanie listą znajomych – użytkownik poprzez podanie ID znajomego, może dodać go do listy znajomych (przechowywanej na jego komputerze). Może również usunąć osobę z listy.
* Przeglądanie listy znajomych – użytkownik może przeglądać listę znajomych, których wcześniej dodał i sprawdzić ich dostępność na serwerze.
* Rozpoczęcie rozmowy – użytkownik wybiera z listy znajomych wszystkie osoby z którymi chce nawiązać połączenie (w szczególności jedną).
* Wysłanie wiadomości – użytkownik wprowadza treść wiadomości w polu tekstowym i zatwierdza jej wysłanie klawiszem ENTER. Wiadomości są szyfrowane.
* Wysłanie pliku – użytkownik wybiera w oknie rozmowy opcje przesłania pliku. W osobnym oknie dialogowym wybiera plik z dysku. Osoba odbierająca plik musi być dostępna oraz wyrazić zgodę na transfer. W momencie przerwania połączenia w trakcie przesyłania pliku, po ponownym podłączeniu plik należy przesłać od początku.
* Odebranie pliku – użytkownik wyraża zgodę na odbiór pliku, po czym w oknie dialogowym wybiera miejsce na dysku komputera, w którym chce zapisać odbierany plik.
  1. Aplikacja serwera – aktor: Serwer
* Uwierzytelnianie użytkowników – serwer odbiera żądanie uwierzytelnienia od klienta. Sprawdza dane w bazie i przesyła z powrotem kod błędu (w szczególności brak błędu).
* Inicjacja połączenia – serwer odbiera żądanie rozpoczęcia rozmowy między użytkownikami, zestawia połączenie i informuje odbiorców o nowej rozmowie.
* Zakończenie połączenia – serwer odbiera żądanie odłączenia się użytkownika od rozmowy. W momencie odłączenia się ostatniej osoby serwer zamyka połączenie.
* Rejestracja nowego użytkownika – serwer odbiera żądanie założenia nowego konta. Sprawdza czy konto o podanym loginie już istnieje, jeśli nie – konto zostaje utworzone, w przeciwnym razie wysyłany jest kod błędu do użytkownika.
* Przesłanie wiadomości – serwer odbiera wiadomość od jednego z użytkowników uczestniczących w rozmowie i przesyła ją do pozostałych uczestników.
* Przesłanie pliku – serwer odbiera żądanie przesłania pliku od jednego z użytkowników uczestniczących w rozmowie i odpytuje pozostałych o zgodę na odebranie pliku. Plik zostaje przesłany tylko do tych użytkowników, którzy wyrazili zgodę.

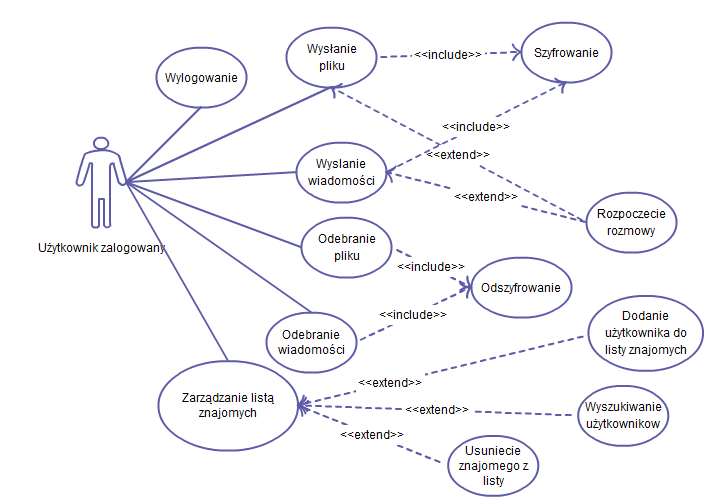
1. Wymagania niefunkcjonalne

* Program działa na systemach operacyjnych z rodziny Linux.
* Program wykonany zostanie z wykorzystaniem framework’u QT (z wyłączeniem klasy QSocket)
* Hasła użytkowników będą szyfrowane, generowanym dynamicznie kluczem, w celu zapewnienia bezpieczeństwa.
* Serwer przyjmuje maksymalną ilość kont użytkowników oraz maksymalną ilość jednoczesnych rozmów jako parametry zewnętrzne.
* Interfejs użytkownika w aplikacji klienckiej będzie intuicyjny i łatwy w obsłudze.

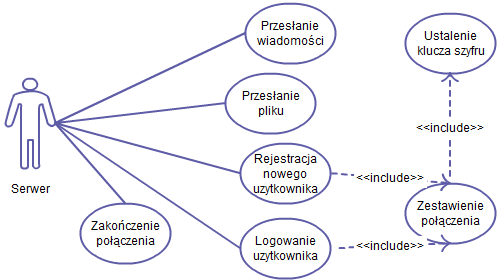
1. Przypadki użycia
   1. Użytkownik niezalogowany



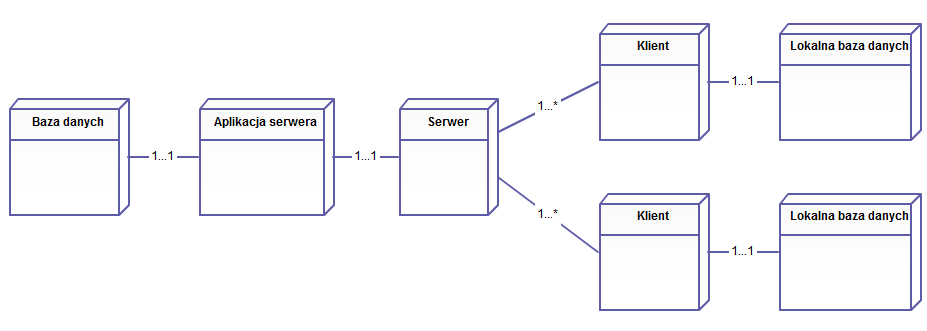
* 1. Użytkownik zalogowany



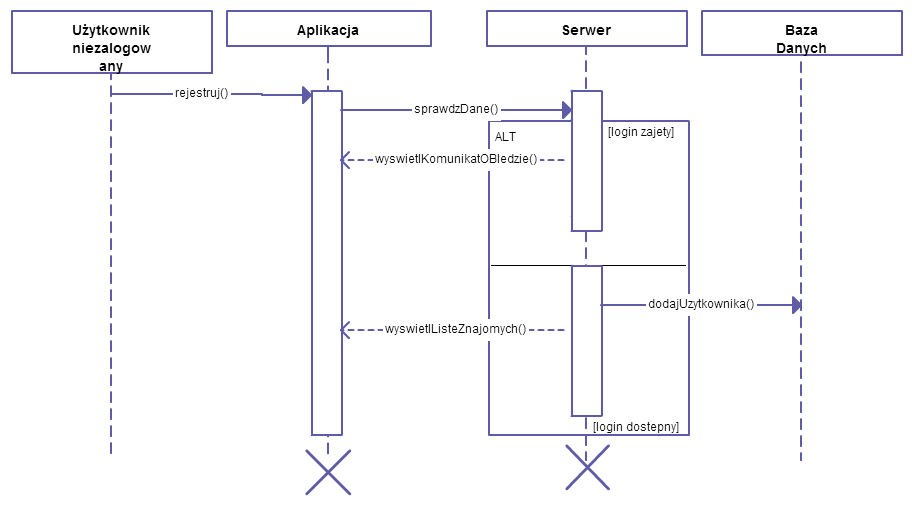
* 1. Serwer



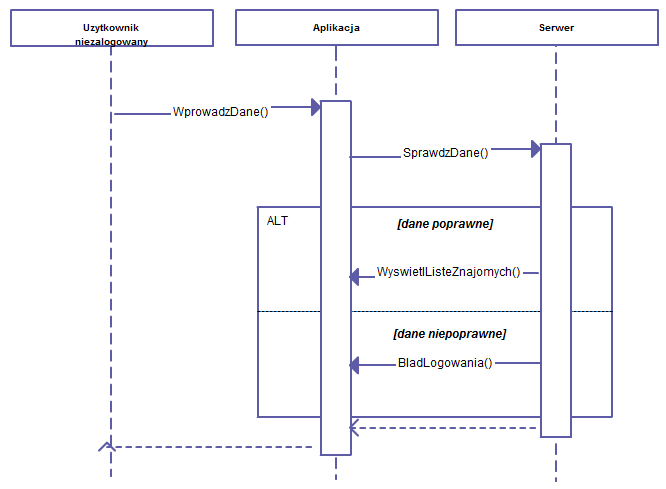
1. Diagram wdrożenia całego systemu



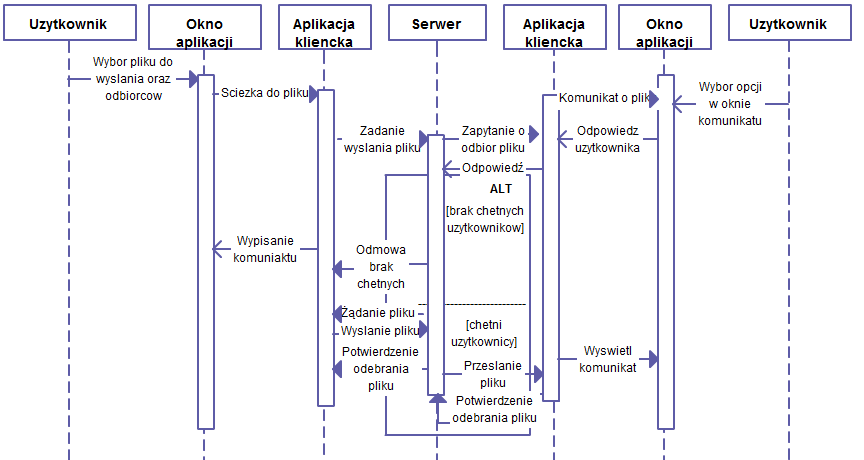
1. Diagramy sekwencji
   1. Rejestracja



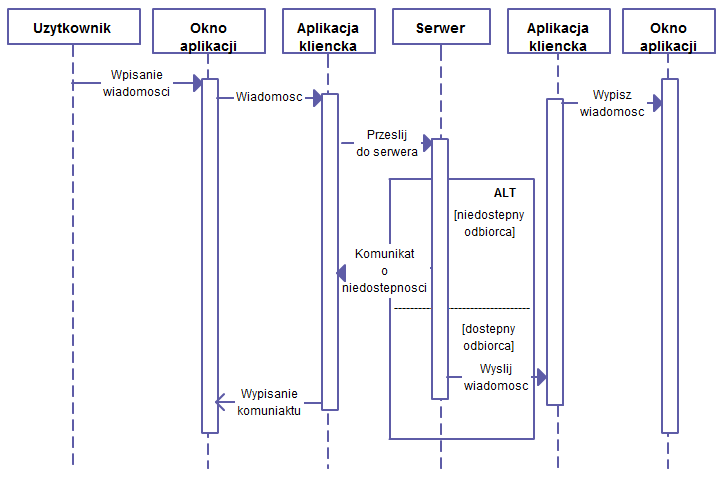
* 1. Logowanie



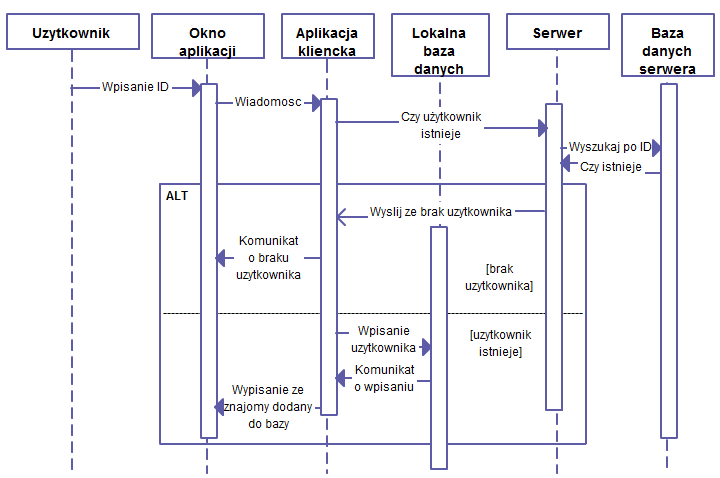
* 1. Wysyłanie/odbiór pliku



* 1. Wysyłanie/odbiór wiadomości



* 1. Dodawanie użytkownika do znajomych



1. 8. Protokół komunikacyjny

Komunikacja odbywała się będzie za pomocą protokołu TCP.

Szkielet wiadomości :

|  |  |
| --- | --- |
| **Nagłówek** | Informacja o ID połączenia oraz typie wiadomości. Zaszyfrowana kluczem do komunikacji klient – serwer. |
| **Dane** | Dane wiadomości – zaszyfrowane kluczem do komunikacji klient – klient. |

* 1. Nagłówek

|  |  |  |
| --- | --- | --- |
| **typ** | int | typ pakietu |
| **ID** | int | ID „wątku” obsługi |
| **rozmiar** | int | całkowity rozmiar danych |

Możliwe typy nagłówków :

* TYP\_PLIK\_ODPYTAJ – odpytaj użytkownik czy chcą odebrać plik
* TYP\_PLIK\_CHETNI – poinformuj nadawcę, ilu jest chętnych
* TYP\_PLIK\_TRANSFER – pakiet z plikiem
* TYP\_REJESTRUJ – rejestracja nowego użytkownika
* TYP\_LOGUJ -

1. 9. Model OSI

|  |  |
| --- | --- |
| **Warstwa aplikacji** | Wysyłanie/ odbieranie wiadomości i plików. |
| **Warstwa prezentacji** | Dołączanie nagłówka do danych (opakowywanie danych). |
| **Warstwa szyfrowania** | Szyfrowanie i deszyfracja danych. |
| **Warstwa sesji** | -- |
| **Warstwa transportowa** | Protokół TCP |
| **Warstwa sieciowa** | -- |
| **Warstwa łącza danych** | -- |
| **Warstwa fizyczna** | -- |

1. 10. API Bazy Danych
   1. Baza serwera

|  |  |  |
| --- | --- | --- |
| **UŻYTKOWNIK** | | |
| **NAZWA** | **TYP** | **OPIS** |
| ID | *int* | Unikatowe ID użytkownika (klucz główny). |
| login | *varchar* | Unikatowy login użytkownika (służy jako nick oraz w trakcie logowania). |
| haslo | *varchar* | Hasło do konta użytkownika. |

* 1. Baza lokalna (aplikacja kliencka)

|  |  |  |
| --- | --- | --- |
| **ZNAJOMI** | | |
| **NAZWA** | **TYP** | **OPIS** |
| ID | *int* | Unikatowe ID użytkownika (klucz główny). |
| login | *varchar* | Unikatowy login znajomego użytkownika. |

* 1. Funkcje (serwer):

**bool sprawdzUzytkownika(int idUzytkownika);** - sprawdza czy użytkownik o podanym ID istnieje (zwraca true jeśli tak)

**int sprawdzUzytkownika(string login);** - sprawdza czy użytkownik o podanym nicku (loginie) istnieje (jeśli tak, zwraca to ID, jeśli nie – zwraca 0)

**string getHashPassword(int idUzytkownika); -** zwraca zhashowane hasło użytkownika

**int dodajUzytkownika (string login, string hash); -** dodaje nowego użytkownika do bazy danych serwera (zwraca 0 jeśli operacja się nie powiodła

* 1. Funkcje (aplikacja kliencka):

**QList getListaZnajomych();** - funkcja zwracająca listę znajomych użytkownika

**bool dodajZnajomego(int idUzytkownika, Qstring nazwaUzytkownika);** - funkcja dodająca do lokalnej bazy danych znajomego (zwraca true jeśli operacja się powiodła).

**bool usunZnajomego(int idUzytkownika);** - funkcja usuwająca użytkownika z lokalnej bazy danych znajomych (zwraca true jeśli operacja się powiodła).

**bool czyWBazie(int idUzytkownika);** - funkcja sprawdzająca po id czy dany znajomy jest już w bazie (zwraca true jeśli tak)

**bool czyWBazie(QString nazwaUzytkownika);** - funkcja sprawdzająca po loginie czy dany znajomy jest już w bazie (zwraca true jeśli tak)

1. 11. Szyfrowanie
   1. Funkcje:

**E\_CODE nawiazPolaczenie (M\_Sock\*); -** funkcja nawiązująca szyfrowaną sesję przez zadany socjet.

**int dajSekret(); -** zwraca klucz szyfrowania.

**string szyfrujDane(string, int); -** funkcja szyfrująca dane na podstawie klucza.

**struct Nag szyfrujNaglowek(Nag\*, int); -** funkcja szyfrująca konkretny nagłówek.

**string deszyfrujDane(Nag\*, int); -** funkcja deszyfrująca dane.

**struct Nag deszyfrujNaglowek(Nag\*, int); -** funkcja deszyfrująca wybrany nagłówek.

**E\_CODE zakonczPolaczenie(); -** funkcja kończąca szyfrowaną sesję.

1. 12. Podział pracy

*Domagała Bartosz* – aplikacja kliencka

*Kaczor Bartosz* – aplikacja serwera

*Łopusiński Paweł* – protokół wiadomości i szyfrowania

*Stepnowski Marcin* – interfejs bazy danych