Paweł Korobczyński 136739

Bartosz Świtaj 136818

Komunikator typu „Gadu-Gadu”

1. Opis protokołu komunikacyjnego
   1. Widomości wysyłane od serwera do klienta (pierwsza cyfra wiadomości wysłanej od serwera zawsze jest typem wiadomości, którą można zinterpretować; do separacji składowych wiadomości zostały użyte „&” oraz „%”):
      1. 6&aa&bb – odpowiedź na próbę rejestracji bądź logowania przez użytkownika, gdzie „aa” oznacza typ wiadomości, do której się odnosi (0 w przypadku rejestracji, 1 w przypadku logowania), a „bb” oznacza powodzenie (1) albo niepowodzenie (0) operacji. Przy pierwszym użytkowniku na serwerze zamiast liczby 6 wysyłana jest liczba 8, aby klient nie oczekiwał wiadomości z listą zarejestrowanych w bazie użytkowników.
      2. 3&-1&a1%b1%c1&a2%b2%c2&…an%bn%cn – wiadomość ze wszystkimi istniejącymi (online/offline) użytkownikami w bazie, którą otrzymuje klient zaraz po pomyślnym zarejestrowaniu lub zalogowaniu o ile nie jest pierwszym użytkownikiem w bazie. „3&-1&” jest to początek wiadomości, który nigdy się nie zmienia (oznacza typ wiadomości oraz id serwera), „an%bn%cn” oznacza kolejno: id użytkownika w bazie, login użytkownika w bazie oraz liczba mówiąca czy jest aktualnie online(1) czy offline(0) .
      3. 5&aa&bb%cc – jest to powiadomienie, które otrzymuje każdy klient będący aktualnie online, mówiące o tym, że powstał nowy użytkownik bądź istniejący już w bazie zmienił status. „aa” oznacza id użytkownika, który zmienia status, „bb” oznacza jego login, a „cc” oznacza jego nowy status (1- online, 0 – offline).
      4. 2&aa&bb – wiadomość, którą dany klient otrzymał od innego użytkownika. „aa” oznacza id nadawcy wiadomości, a „bb” oznacza treść wiadomości.
   2. Wiadomości wysyłane od klienta do serwera
      1. 0&aa – prośba o zarejestrowanie, gdzie „aa” to liczba określająca długość kolejnej wiadomości wysłanej przez danego klienta, zwierającej strukturę *login&hasło*
      2. 1&aa – prośba o zalogowanie do systemu, gdzie „aa” to liczba określająca długość kolejnej wiadomości wysłanej przez danego klienta, zwierającej strukturę *login&hasło*
      3. 2&aa – prośba o przesłanie wiadomości do innego użytkownika, gdzie „aa” to liczba określająca długość kolejnej wiadomości wysłanej przez danego klienta, zwierającej strukturę *id\_do\_kogo&wiadomość*
2. Opis implementacji
   1. Serwer

Po kompilacji wszystkich plików serwerowych, przy uruchamianiu serwera jest możliwość podania portu na jakim ma działać serwer. Serwer składa się z pliku „main.cpp”, który jest głównym plikiem projektowym i tworzy wszystkie potrzebne pliki do uruchomienia serwera. Klasa „src/models/Server.cpp” odpowiada za działanie całego serwera. Do konstruktora klasy przekazywany jest port, który został podany na wejście programu uruchomieniowego.

Najważniejsze metody znajdujące się w klasie Server.cpp:

- setup() odpowiada za całą konfigurację serwera (utworzenie gniazda, bindowanie gniazda itd.),

- start() odpowiada za uruchomienie głównej pętli nasłuchującej i odbierającej przychodzące połączenia, dla każdego połączenia wywołuje metodę handleConnection(), a ta tworzy nowy wątek. Dodatkowo tworzy jeden wątek odpowiedzialny za wysyłanie wiadomości do klientów,

- readFromSingleClientThread() – metoda, która funkcją nowego wątku, tworzonego dla każdego klienta. Czeka ona najpierw na nadesłanie nagłówka wiadomości z typem oraz długością wiadomości docelowej, a następnie odczytuje wiadomość docelową i wykonuje odpowiednie operacje np. rejestracja nowego użytkownika. Wątek ten korzysta z wielu dostępnych metod do wykonania odpowiednich operacji np. registerUser(), loginUser(), createMessage(), createResponesMessage() itd. Po rozłączeniu połączenia z klientem wątek zostaje zakończony,

- sendToAllClientsThread() – metoda obsługująca wątek odpowiedzialny za wysyłanie wiadomości do wszystkich klientów. Każda wiadomość, która została stworzona na serwerze bądź odebrana od klienta trafia do wektora o nazwie messages, który przechowuje obiekty klasy Message. Wątek odpowiedzialny za wysyłanie wiadomości sprawdza po kolei każdą wiadomość znajdującą się w wektorze (np. czy nadawca jest online) i jeśli nie ma przeciwskazań wysyła ją w odpowiedni sposób do nadawcy.

Do przechowywania zarejestrowanych użytkowników w bazie wykorzystany został wektor users, który przechowuje obiekty klasy User, a ta posiada wszystkie potrzebne informacje na temat klienta.

Dodatkowo wykorzystane zostały dwa mutexy usersMutex i messagesMutex, które pozwalają na współbieżne korzystanie z wektorów z obiektami.

* 1. Klient

Po uruchomieniu aplikacji klienta, uruchomiony zostaje wątek odpowiedzialny za obsługę graficzną okna logowania,tworzony przez bibliotekę JavaFX. Ten sam wątek odpowiada za wysłanie wiadomości do serwera w celu zalogowania lub zarejestrowania się do systemu. Po otrzymaniu odpowiedzi o sukcesie logowania, wątek ten otrzymuje od serwera informację o wszystkich zalogowanych użytkownikach i rozpoczyna graficzną obsługę listy tych użytkowników. Tworzy przy tym dodatkowy wątek, który odbierać będzie wszystkie informacje od serwera na temat zmiany statusu użytkowników oraz nowych wiadomości od innych użytkowników i będzie zapisywać wszystkie konwersacje w utworzonej wcześniej strukturze. Jest to lista obiektów utworzonej przez nas klasy User, które posiadają między innymi pole typu StringBuffer, zapisujące całą konwersację. Po wybraniu użytkownika z listy i kliknięciu przycisku „Open chat” uruchomiony zostanie nowy wątek JavaFX, odpowiedzialny za wysyłanie informacji do serwera o nowej wiadomości oraz pomocniczy wątek, który odświeża pole, w którym pojawiać się będzie przesyłana konwersacja.

1. Opis sposobu kompilacji i uruchomienia projektu
   1. Po pobraniu projektu należy otworzyć katalog gg-master/ggServer
   2. W terminalu należy wykonać następującą komendę w celu skompilowania serwera:

*g++ -Wall -pthread main.cpp src/models/Server.cpp src/models/Message.cpp src/models/User.cpp -o main*

* 1. Po zakończonej kompilacji należy wykonać komendę *./main* w celu włączenia serwera (można również podać port, na którym zostanie uruchomiony serwer; opcjonalnie -1234)
  2. Następnie, można uruchomić klienta, włączając wykonywalny plik ggClient.jar znajdujący się w katalogu gg-master/ggClient/out/artifacts/ggClient/