Sprawozdanie - Communicator

Autorzy: Nikolas Szwargot 151735
Mikołaj Napierała 151812

I. Temat

Komunikator internetowy. Aplikacja pozwalająca na komunikację między klientami. Wymagane: komunikator ma pozwalać na tworzenie konta, logowanie z hasłem, wyświetlenie znajomych, prowadzenie wielu konwersacji jednocześnie. Dane użytkowników zapisywane poza programem. Wymagane GUI.

II. Sposób wykonania

Zgodnie z tematem projekt został wykonany w architekturze klient - serwer w dwóch głównych plikach: server.cpp (kod serwera, wykonany używając C++ i API bsd-sockets) i index.html (Kod klienta, napisany w HTML i JavaScript, korzystający również z pliku styles.css, który jest zawarty w projekcie.

III. Kompilacja i uruchomienie programu

1. **Kod znajduje się w repozytorium GitHub pod linkiem: GitHub**
2. Pliki właściwe znajdują się dla serwera: server/server.cpp oraz dla klienta client/index.html.
3. Do kompilacji kodu C++ wykorzystaliśmy CMake.
4. Opis kompilacji znajduje się w pliku README.md. (po skopiowaniu repozytorium należy zainicjować submudules)
5. **Kompilacja serwera:** (w katalogu z plikiem CMakeLists.txt oraz server.cpp)
   cmake .
   make
6. **Uruchomienie serwera (w katalogu serwer):**
   ./Communicator
7. **Po uruchomieniu serwera można korzystać z komunikatora. Aby uruchomić komunikator, należy w dowolnej wyszukiwarce otworzyć plik index.html.**

IV. Opis techniczny projektu

1. Wykorzystane technologie:
   a. Serwer:
      a) Napisany z wykorzystaniem języka C++
      b) API bsd-sockets do komunikacji z klientami
      c) pthread do tworzenia nowych wątków dla przychodzących zapytań
      d) nlohmann json (zaimportowane jako submodule na github) - biblioteka udostępniająca strukturę danych, którą wykorzystaliśmy zapytań oraz do przechowywania danych.
      e) fstream - standardową bibliotekę do zapisu i odczytu danych z pliku
   b. Klient
      a) Struktura strony napisana w HTML z wykorzystaniem CSS
      b) JavaScript do obsługi wysyłania zapytań oraz odbierania danych.

2. Struktura aplikacji:
    a. Zaimplementowaliśmy podstawowe REST API do komunikacji klienta z serwerem.
    b. Każde przychodzące zapytanie jest delegowane na osobny wątek, który się kończy po wysłaniu wykonaniu odpowiednich działań.
    c. Zabezpieczenie pamięci mutexami do wykluczenia jednoczesnego zapisu oraz odczytu z plików.

V. Sposób korzystania z programu

Po otwarciu pliku index.html i uruchomieniu serwera, należy zarejestrować konto po wpisaniu nazwy użytkownika w pole username, hasła w pole password i naciśnięciu przycisku register. Następnie po rejestracji, należy wpisać dane ponownie aby zalogować się do serwisu.

Po poprawnym zalogowaniu zamykamy komunikat o sukcesie operacji. Aby dodać znajomego należy wybrać przycisk “Add friend”, wybrać użytkownika z listy lewym przyciskiem myszy i kliknąć przycisk “Add selected friend”.

Po dodaniu użytkownika do listy znajomych, możemy do niego wysłać wiadomość po wybraniu go z listy “Friends list”. Aby ten użytkownik odebrał od nas wiadomość, musi również dodać nas do znajomych.Jeśli to zrobi może do nas wysłać wiadomość, a my możemy ją od razu odebrać! Aby otworzyć drugi chat, nie zamykając poprzedniego możemy otworzyć nową kartę, zalogować się na wybrane konto i otworzyć wybrany chat.

V. Konta do testowania:

Aby przetestować aplikację stworzyliśmy trzy przykładowe konta:

1) Login: test Hasło: testpassword
2) Login: niko Hasło: nikopassword
3) Login: miki Hasło: mikipassword

VI. Zasada działania

- Serwer zapisuje i sprawdza loginy, hasła oraz listy znajomych w pliku usersinfo.json, tak też sprawdza rejestrację i logowanie.
- Wiadomości również są zapisywane w pliku json (messages.json). Po otwarciu czatu serwer wysyła filtrowaną historie wiadomości do klienta.
- Czaty są odświeżane z prośbą o przesłanie nowych wiadomości co jedną sekundę.