Sprawozdanie – opis projektu

Protokół komunikacji

Komunikacja między serwerem a klientem odbywa się przy pomocy komunikatów tekstowych w formacie JSON. Przed nadaniem właściwego komunikatu, zarówno serwer, jaki i klient, wysyłają informacje o rozmiarze wysyłanych danych. Jest to wysyłany i czytany bajt po bajcie ciąg tekstowy w postaci #liczba_bajtów#. Druga strona po odebraniu tej wiadomości oczekuje na nadchodzący komunikat - czyta dane w pętli aż do otrzymania kompletnej wiadomości (czyli do momentu, gdy funkcja write odczyta spodziewaną liczbę bajtów, być może we fragmentach). Wysyłanie także odbywa się w pętli, aż do momentu, gdy cały komunikat zostanie nadany z powodzeniem.
Odzwierciedlana przez komunikaty struktura danych jest zdefiniowana po stronie serwera w języku c++:

```
typedef struct message
{
   int type;
   int userID;
   char title[TITLE_SIZE];
   char username[USERNAME_SIZE];  // author
   char msg[MESSAGE_SIZE]; // content
   char tags[TAGS_SIZE];
} message;
```

Pole type określa rodzaj komunikatu. Jest to wartość liczbowa interpretowana w następujący sposób:

0 - user authentication initialization from client
1 - user authentication response to client (success: send assigned userID)
2 - user authentication response to client (error: user already logged in)
100 - disconnect notification from client to server

10 - user text message (post to be sent to subscribers)

20 - client request to subscribe user
21 - server response: subscribe success
22 - server response: subscribe fail: user already subscribed or does not exist
23 - server response: subscribe fail: user cannot subscribe himself

30 - client request to unsubscribe user
31 - server response: unsubscribe success
32 - server response: unsubscribe fail: user was not subscribed or does not exist
33 - server response: subscribe fail: user cannot unsubscribe himself

Po przyłączeniu klienta serwer oczekuje komunikatu typu 0 z niepustym polem username (login). Jeżeli logowanie powiedzie się, odsyłany jest typ 1 z nadanym userID.

Klient zapamiętuje userID i wysyła je w każdym kolejnym komunikacie, podobnie jak swój login w polu username.
W komunikacie typu 10 w polu msg umieszczona jest wiadomość użytkownika. Serwer rozsyła ją do odbiorców.

W komunikacie typu 20 (30) w polu msg umieszczona jest nazwa użytkownika do subskrybowania (anulacji subskrypcji). Serwer analizuje i odpowiada na żądanie:
Odpowiedź typu 21 (31) oznacza sukces operacji.
Odpowiedź typu 22 (32) oznacza błąd: podany użytkownik albo nie istnieje, albo jest już subskrybowany (nie był subskrybowany).
Odpowiedź typu 23 (33) oznacza błąd: użytkownik próbował subskrybować (anulować subskrypcję) samego siebie.

Opis implementacji serwera

Serwer wielowątkowy napisany w języku c++11 przy użyciu API BSD Sockets. Po inicjalizacji socketu serwer w pętli nasłuchuje połączeń od klientów. Akceptując klienta tworzy dla niego osobny wątek do obsługi jego żądań. Wątek w pętli odczytuje komunikaty przychodzące (read), interpretuje je zgodnie z wartością pola type (wg. Protokołu komunikacji), wykonuje żądane operacje i odsyła odpowiedź (write) do klienta. Serwer przechowuje informacje o zalogowanych użytkownikach w tablicy struktur userData, a informacje o deskryptorach przyłączonych klientów w tablicy connectionSocketDescriptors. Dane te są przekazywane wątkowi poprzez indywidualną dla każdego wątku strukturę threadData. Dostęp do danych w sekcjach krytycznych jest kontrolowany poprzez pthread mutex'y, co umożliwia poprawność działań współbieżnych serwera. Serwer weryfikuje poprawność żądań klienta i informuje klienta o błędzie/poprawności wykonania operacji. Komunikaty serwer-klient tworzone są jako struktura message, a następnie transformowane do reprezentacji tekstowej w formacie JSON i wysyłane do klienta (i odwrotnie w przypadku odbierania komunikatu). Gdy klient kończy połączenie, serwer wykrywa zwracaną przez funkcję read wartość równą 0. Wątek klienta jest kończony, a związane z nim zasoby są wówczas zwalniane.

Opis implementacji klienta

Klient oparty jest o framework Electron, który umożliwia pisanie natywnych aplikacji na różne platformy z wykorzystaniem technologii webowych. Aplikacja jest dwuwątkowa, gdzie wątek główny prowadzi komunikację z serwerem, a wątek pomocniczy jest odpowiedzialny za obsługę interfejsu użytkownika. Wątek główny wykorzystuje bibliotekę „net" platformy node.js, która implementuje gniazda sieciowe, dzięki czemu możliwe jest bezpośrednie połączenie między serwerem, a klientem. Komunikacja z serwerem polega na wymianie plików w formacie JSON o polach opisanych w sekcji „Protokół komunikacji". API umożliwiające komunikację między wątkami jest częścią Electrona i pozwala na asynchroniczny oraz wielokanałowy przepływ komunikatów, co zapewnia płynność działania interfejsu.

Sposób kompilacji i uruchomienia

Aplikację serwera należy skompilować własnoręcznie, postępując według następujących kroków:

1. Należy sklonować repozytorium https://github.com/NiewidzialnyCzlowiek/pubsub-webapp lub pobrać kod z najnowszego wydania.
2. W repozytorium znajduje się folder „server”, w którym należy uruchomić komendę make. Przygotowany przez nas plik makefile gwarantuje poprawną kompilację programu.

Po kompilacji serwer można uruchomić wpisując w terminalu komendę „./serverJ [numer_portu]”, gdzie numer_portu jest portem, na którym serwer będzie przyjmował połączenia od klientów. W przypadku pominięcia tego parametru przy uruchomieniu, serwer użyje domyślnie portu 1234.

W celu uruchomienia klienta należy pobrać odpowiedni pakiet udostępniony w najnowszym wydaniu z repozytorium. W wydaniu 2.0.0 udostępniliśmy pakiet dla systemów macos, linux 64bit oraz linux 32bit. W systemach linux, po pobraniu pliku, należy ustawić go jako wykonywalny, a następnie uruchamiać jak inne aplikacje.