Offline Messenger

Daniel Apopei

Universitatea Alexandru Ioan Cuza Iasi

1 Introducere

1.1 Prezentare Generala

Aceasta aplicatie va permite utilizatorilor sa comunice intre ei prin intermediul unor comenzi predefinite.

Utilizatorii vor putea sa primeasca mesajele instant de la alti utilizatori, in cazul in care sunt online.

Daca nu sunt online, aplicatia ofera si un istoric al conversatiei pentru fiecare pereche de utilizatori.

De asemenea, ei pot raspunde la anumite mesaje pentru a oferi partenerului de conversatie context pentru anumite mesaje.

1.2 Lista Functionalitati

1. Sistem de Logare / Delogare: conturile vor fi protejate cu o parola pe care userii o vor introduce la inregistrare sau de fiecare data cand intra pe aplicatie.
2. Vizualizare Conversatii: pentru organizarea mesajelor, fiecare conversatie va avea pagina ei.
3. Vizualizare Conversatie: utilizatorul va intra pe o anumita conversatie, care ii va deschide tot istoricul mesajelor din acea conversatie.
4. Trimitere Mesaj: utilizatorul poate adauga un nou mesaj. daca partenerul de conversatie e online si deschide aceeasi conversatie, atunci mesajul va aparea instant pe ecranul lui.
5. Raspundere Mesaj: similar cu Trimitere Mesaj, dar acest mesaj va include si textul mesajului la care se raspunde.

2 Tehnologii Aplicate

- Comunicare prin Socket-uri TCP: in cazul trimiterii mesajelor, prioritatea este corectitudinea datelor. In consecinta, alegerea TCP e evident superioara UDP.
- Stocare in Baze de Date: este necesara conservarea datelor chiar si dupa ce aplicatia server se inchide (uneori din motive obiectve, ex: pana de curent). In contrast cu fisiere JSON / XML, e necesar un mecanism de comunicare pentru evitarea "race condition". Insa, pentru bazele de date (SQLite), nu e necesar deoarece sistemul asigura aceasta. Mai mult, constrangerile ce pot fi setate la crearea tabelelor din baza de date, ofera un mecanism de verificare a validitatii datelor de la intrare.

3 Structura Aplicatiei

Aplicatia consta dintr-o aplicatie server si o aplicatie client.

In figura 1, putem vedea un exemplu de comunicare.

Inainte de pornirea clientului, e necesar ca serverul sa ruleze. Clientul se va conecta automat la el, apoi asteptand comenzi de la utilizatori.

Clientul nu va comunica cu baza de date si nici nu va executa calcule (cu exceptia validarii sintaxei comenzilor si a parametrilor). Toate aceste sarcini ii revin serverului.

4 Aspecte de Implementare

4.1 Thread-uri

Se pune problema receptionarii mesajelor instant de catre client. Totodata, acesta trebuie sa preia input de la tastatura. Cum ambele operatii sunt blocante, e necesara folosirea a mai multor thread-uri. In cazul acesta, doua sunt suficiente:

- receiveThread: va asculta pe socket pentru mesaje de la server. odata ce primeste mesaj, va extrage si va afisa pe ecran informatia relevanta.
- userInputThread: va citi de la tastatura comenzi si le va formata astfel incat sa poata fi transmise catre server intr-o maniera uniforma

Pentru server, exista mai multe optiuni:

- mai multe procese (tata-fiu): e necesar un mecanism de comunicare intre procese, care creste complexitatea mai mult decat e nevoie
- mai multe thread-uri: e necesar un mecanism de evitare "race condition" prin lacate mutex
- un singur thread.

Folosim un singur thread impreuna cu select pentru a asculta pentru modificari in toate socket-urile.

De asemenea, folosim un vector de tip Connection, pentru a tine cont cati si ce clienti sunt conectati si daca si cu ce nume de utilizator sunt logati fiecare. Astfel, putem determina la ce tip de comenzi / mesaje au acces. Totodata, putem sti, daca e cazul sa trimitem un MESSAGE_NOTIFICATION catre acel socket, daca clientul vizioneaza la un moment dat o conversatie anume (prin currentView si viewingConvo).

Listing 1.1. Structura Packet

```
struct Connection {
    int sd;
    char username[USERNAME_LENGTH];
    ViewType currentView;
    char viewingConvo[USERNAME_LENGTH];
};
```

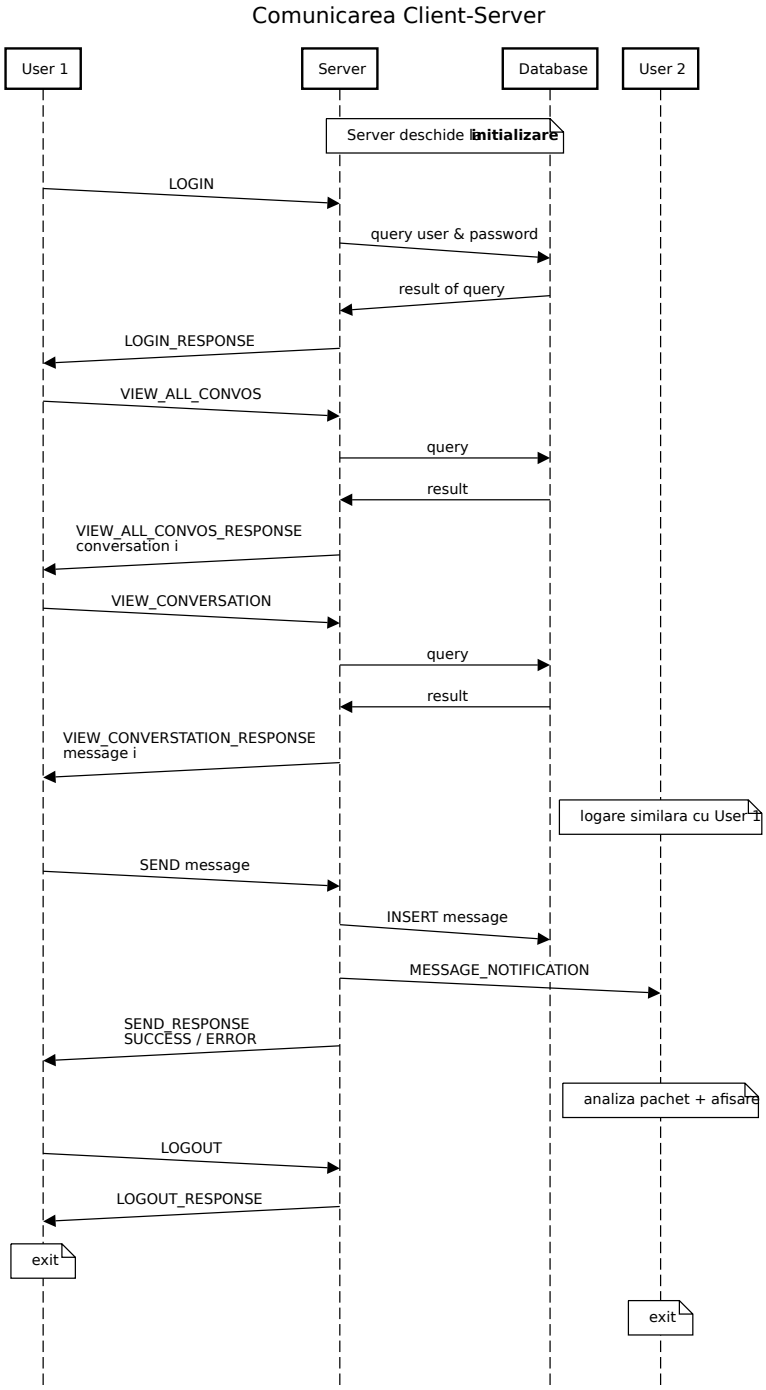

Fig. 1. Your Diagram

4.2 Protocolul de comunicare

Pentru a evita eventuale bug-uri la asamblarea / dezasamblarea unor string-uri, folosim structuri.

Listing 1.2. Structura Packet

```
struct Packet {
    PacketType type;
    ErrorType error;
    User user;
    Message message;
};
```

Aceasta e singura structura trimisa pe socket. Nu toate campurile vor fi folosite la orice comanda. Spre exemplu, la LOGIN se va folosi doar campul user. Clientul va pune in user.username si user.password informatia preluata de la tastatura si va specifica in campul type tipul de pachet, pentru ca serverul sa stie ce sa faca cu pachetul. Iata tipurile de pachete posibile:

Listing 1.3. Tipuri de pachet

```
enum PacketType {
    EMPTY,
    REGISTER,
    REGISTER_RESPONSE,
    LOGIN,
    LOGIN_RESPONSE,
    LOGOUT,
    LOGOUT_RESPONSE,
    SEND_MESSAGE,
    SEND_MESSAGE_RESPONSE,
    MESSAGE_NOTIFICATION,
    VIEW_ALL_CONVOS,
    VIEW_ALL_CONVOS_RESPONSE,
    VIEW_CONVERSATION,
    VIEW_CONVERSATION_RESPONSE
};
```

Pe baza tipului de pachet, serverul va verifica / altera baza de date si va returna un pachet de tip RESPONSE in care va avea campul error SUCCESS sau un cod de eroare.

Listing 1.4. Tipuri de erori

```
enum ErrorType {
    SUCCESS,
    USER_ALREADY_EXISTS,
    INVALID_USER_DATA,
    USER_ALREADY_CONNECTED,
```

```
    NOT_LOGGED_IN,
    NOT_LOGGED_OUT,
    INVALID_REPLY_ID
};
```

4.3 Scenarii

Login

- user tasteaza LOGIN user1 pass
- client trimite Packet cu type LOGIN
- server analizeaza si extrage din pachet informatia
- server trimite query la baza de date
- daca gaseste un rand, inseamna ca exista, deci updateaza array-ul connectionList si intoarce clientului un Packet LOGIN_RESPONSE SUCCESS

View Conversation

- initial, chiar daca e logat, userul e pe MAIN_VIEW (similar cu ecranul de start din Facebook Messenger, lista de conversatii)
- asta inseamna ca nu primeste niciun mesaj instant, insa mesajele vor fi totusi introduse in baza de date
- daca user tasteaza VIEWCONVO user2, se schimba view si daca user2 ii trimite mesaj va primi instant
- pachete: VIEW_CONVO, VIEW_CONVO_RESPONSE

Reply Message

- user tasteaza reply 23 hello world
- server trimite query catre baza de date si verifica daca mesajul cu id-ul 23 exista si daca combinatia (sender, receiver) e aceeasi cu conversatia
- daca nu, returneaza pachet INVALID_REPLY_ID
- daca da, atunci apendeaza la content-ul mesajului, content-ul mesajului cu id-ul 23. astfel mesajul va avea continut atasat, persistent in baza de date.

5 Concluzii

- se poate implementa o metoda de criptare / decriptare in client (oferind end-to-end encryption)
- se poate implementa criptare / decriptare in server pentru parolele introduse in baza de date

6 Referinte Bibliografice

References

1. UAIC Computer Networks Page, https://profs.info.uaic.ro/ computernetworks/index.php
2. https://staff.fmi.uvt.ro/ stelian.mihalas/com_net/courses/sockets.pdf