Documentaţie Championship

Morcov-Pahoncea Răzvan, Anul 2, Grupa A5

Universitatea "Alexandru Ioan Cuza" din Iași

1 Introducere

Championship este o aplicație de tip client/server pentru management de competiții de tip *E-Sports*. Oferă utlizatorilor două tipuri de conturi (*regular* și *admin*), fiecare putând controla diferite aspecte ale competițiilor (utilizatorii *regular* se pot conecta, înscrie la competiții, dar pot opta și pentru schimbarea datei competițiilor, cu acordul unui user de tip *admin*, cei din urmă putând crea competiții, stabilind regulile care trebuie respectate, dar și orele și datele la care au loc). Interfața aplicației este un terminal *Linux*, care nu este foarte *user-friendly*, oferind totuși comenzi simple și utile pentru funcționalitățile menționate.

2 Tehnologii utilizate

Pentru schimbul de informații dintre utilizatori (clienți) și server, aplicația *Championship* folosește *TCP (Transmission Control Protocol)*. Această tehnologie este utilizată de aplicații care necesită confirmări de primire a datelor. Efectuează o conectare virtuală full duplex între două puncte terminale, fiecare punct fiind definit de o adresă IP și de către un port TCP[1]. Implementarea folosește un model de concurență al clienților pentru modelul TCP, astfel fiind posibilă conectarea unui număr nelimitat de utilizatori la un moment dat.
Aplicația necesită siguranța faptului că datele sunt transmise între client și server în ordinea în care sunt trimise, lucru pe care protocolul TCP îl asigură. Astfel că utilizatorul are o experiență plăcută și are garanția că informațiile pe care le transmite către server ajung numai la acesta, primind confirmare și răspuns pentru fiecare request pe care îl face.
Aplicația este și destul de eficientă din punct de vedere al resurselor utilizate, deoarece după satisfacerea tuturor request-urilor de la utilizatori și finalizarea tuturor transferurilor de date, serverul închide conexiunea, eliberând astfel resursele.
Protocolul TCP a fost ales pentru siguranța ordinii datelor, fiind vorba despre un *login/register* înainte de procesarea altor request-uri, dar și datorită faptului că este suficient de *lightweight* din punct de vedere al implementării și rulării, necesitând destul de puține resurse pentru fiecare conexiune.

3 Arhitectura aplicației

Aplicația *Championship* este structurată în minim două executabile (un server și minim un client). Executabilul client asigură utilizatorului o interfață prin care

poate comunica cu server-ul pentru a-i trimite request-uri. Executabilul server reprezintă *endpoint-ul* unde un client (sau mai mulți) se poate conecta. Server-ul primește request-ul de la client, îl procesează, identifică, rezolvă și trimite înapoi răspunsul către client, care îl afișează pe ecran utilizatorului.
Procesarea request-urilor se face de către server prin extragere, inserare, update dintr-o bază de date *SQLite3*, dar și prin trimiterea de e-mailuri folosind *libcurl*.

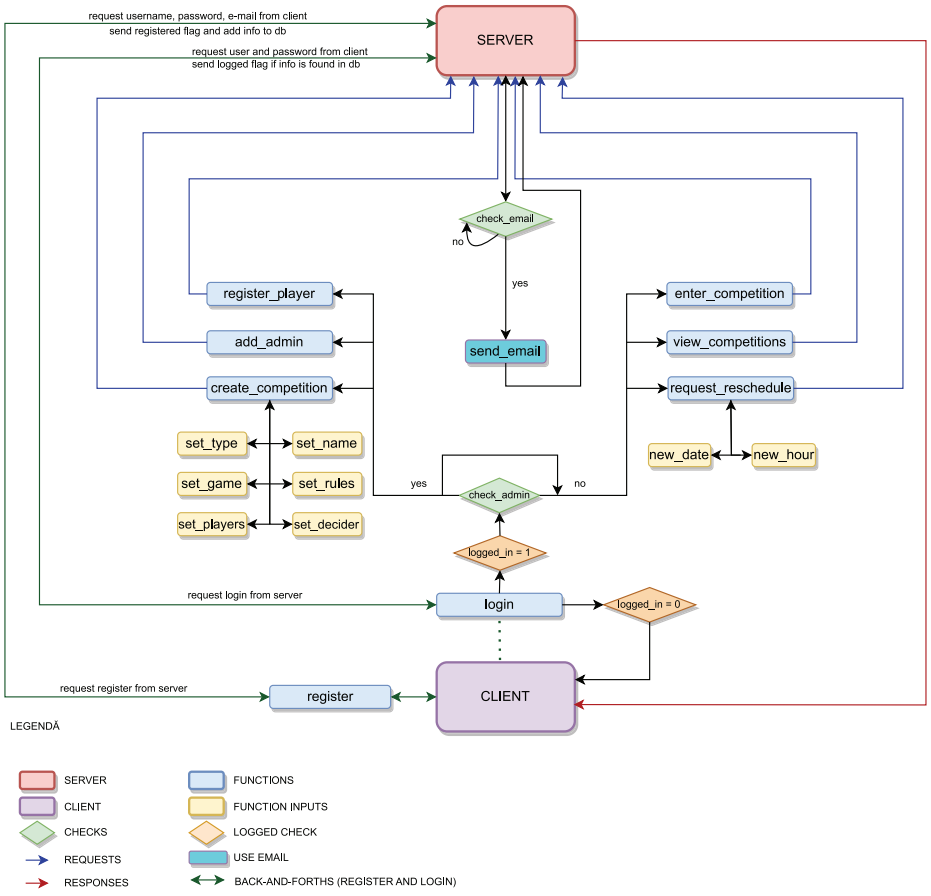

Fig. 1. Diagramă de flow
Notă. Toate răspunsurile de la server ajung în client, acesta le afișează pentru utilizator, apoi trece peste secțiunea de *login* sau *register* și continuă execuția cu utilizatorul curent logat. (de unde și linia punctată dintre *CLIENT* și *login*)

După cum se poate observa din diagrama de mai sus, există două tipuri de utilizator (regular și administrator), fiecare având capabilități diferite. Crearea

unui cont se poate face cu *register*, iar conectarea, mai apoi, cu *login*, pentru fiecare dintre aceste request-uri primindu-se apoi prompt-uri pentru client pentru a introduce informațiile necesare pentru procesarea în totalitate a comenzii. Administratorii pot adăuga campionate, pot alege tipul lor, numărul de jucători, reguli care trebuie respectate. Utilizatorii pot alege să se înscrie la unul din campionatele create, iar în cazul în care mai sunt locuri și sunt acceptați, să schimbe eventual data sau ora la care sunt repartizați de către server.
Primitivele de bază folosite în aplicație sunt:

- socket() - crează un nou punct terminal al conexiunii[2]
- bind() - atașează o adresă locală la un socket[2]
- listen() - permite unui socket să *asculte* până când primește o conexiune[2]
- accept() - blochează apelantul până la sosirea unei cereri de deconectare (utilizată de serverul TCP)[2]
- connect() - tentativă activă de stabilire a conexiunii (utilizată de clientul TCP)[2]
- write() - scrie informații în socket[2]
- read() - citește informații din socket[2]
- close() - eliberează conexiunea (închide un socket)[2]

4 Detalii de implementare

4.1 Implementare

Aplicația *Championship* funcționează pe baza diagramei prezentate în secțiunea 3 de mai sus. În cele ce urmează vom detalia o parte dintre particularitățile de implementare a câtorva dintre funcțiile prezentate în diagramă.

```
else if (strncmp(received, "register", 8) == 0)
{
    char userInfo[1000];
    /*...
    create userInfo string by requesting information from client
      for every field (name, pass, email, account type)
    ...*/
    // request example:
    write(client, "[Server] Please provide an account type (admin/ regular): ", strlen("[Server] Please provide an account type (admin/ regular): "));
    bzero(received, 1000);
    read(client, received, sizeof(buffer));
    if (strncmp(received, "admin", 5) == 0)
    {
        write(client, "[Server] Account created successfully.\n[Server] Enter command: ", strlen("[Server] Account created successfully.\n[Server] Enter command: "));
        strcat(userInfo, "0);");
        rc = sqlite3_exec(db, userInfo, callback1, 0, &zErrMsg);

        if (rc != SQLITE_OK)
        {
            printf("[Server] DB Error. Server will shut down.\n");
            close(client);
            return 1;
        }
        else
        {
            printf("[Server] User information saved successfully.\n");
            fflush(stdout);
        }
    }
    else if (strncmp(received, "regular", 7) == 0)
    {
        write(client, "[Server] Account created successfully.\n[Server] Enter command: ", strlen("[Server] Account created successfully.\n[Server] Enter command: "));
        strcat(userInfo, "1);");
        rc = sqlite3_exec(db, userInfo, callback1, 0, &zErrMsg);

        if (rc != SQLITE_OK)
        {
            printf("[Server] DB Error. Server will shut down.\n");
            close(client);
            return 1;
        }
        else printf("[Server] User information saved successfully.\n");
```

În porțiunea de cod de mai sus este prezentată funcționalitatea de *register* a unui utilizator, acestuia fiindu-i prezentat câte un prompt pentru fiecare informație pe care trebuie să o introducă (*nume, parolă, adresă e-mail, tip cont*), datele fiind apoi validate și introduse în baza de date *SQLite3*. Dacă introducerea datelor se face corect, atunci clientul va primi promptul *Account created successfully.*. În caz contrar, acesta va primi promptul *Account not created. Please enter another command or try again.*. De asemenea, și watcher-ul serverului va primi informații despre introducerea datelor cu succes (sau nu) în baza de date, pentru a putea efectua *debug* în cazul în care operația *INSERT* nu se face cu succes.

Introducerea datelor în baza de date se face cu ajutorul unei funcții *callback*, dar care pentru operația *INSERT* are body-ul gol, returnând 0 (deoarece validarea datelor este efectuată deja, iar baza de date este cu siguranță deschisă, datorită unui check - a se vedea imaginea de mai jos).

```
sqlite3 *db;
char *zErrMsg = 0;
int rc;
char *sql;
// open db
rc = sqlite3_open("userinfo.db", &db);

if (rc)
{
    printf("[Server] Error on opening database. Server will shut down.\n");
    return 0;
}
else
{
    printf("[Server] Connected to the database successfully.\n");
    fflush(stdout);
}
```

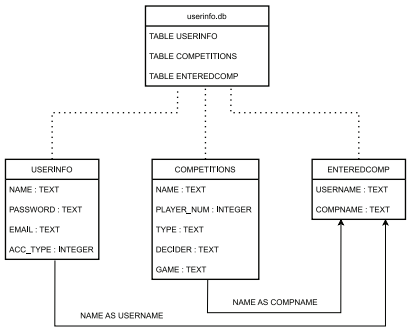

Fig. 2. Database diagram

4.2 Scenariu de utilizare

Un utilizator deschide aplicația, și este întâmpinat de prompt-ul de introducere a unei comenzi, fiindu-i disponibile numai *login* și *register*, toate celelalte fiind blocate de acest *login check*. După crearea unui cont sau conectare, devin disponibile toate celelalte comenzi.
Un utilizator *regular* se conectează și primește prompt de la server că a fost efectuată cu succes conectarea. Apoi, primește din nou prompt-ul de introducere a unei comenzi. Acesta alege să introducă *view-competitions*. Server-ul primește request-ul, îl procesează și întoarce către client o listă cu toate campionatele (numerotate de la 1 la n), apoi oferă un prompt prin care întreabă utilizatorul dacă vrea să se înscrie la vreunul dintre campionatele afișate. În caz afirmativ, îi este cerut să introducă identificatorul (numărul din listă) al campionatului la care vrea să se înscrie. Server-ul primește acest request, verifică dacă utilizatorul este eligibil pentru a se înscrie la campionat (dacă nu s-a atins deja numărul maxim de jucători care pot participa la campionatul respectiv). În cazul în care înscrierererea se poate efectua cu succes, întoarce acest răspuns către client, calculează data și ora la care să participe utilizatorul și trimite un request către server-ul de e-mail prin care să îi transmită utilizatorului detaliile despre campionat. Utilizatorul primește un prompt prin care să introducă o nouă comandă. Acesta alege *logout* și mai apoi *exit*, sesiunea încheindu-se.
La primirea e-mailului, utilizatorul observă că data și ora la care este înscris nu se încadrează în programul său. Astfel că efectuează o nouă conectare și introduce comanda *request-reschedule*. Server-ul primește acest request, îl procesează și trimite înapoi către utilizator un prompt prin care îi cere în formatul *dată:oră* noua dată la care ar putea să participe la campionat. După ce primește răspunsul de la utilizator, server-ul verifică dacă data este validă și dacă nu este deja ocupată de un alt participant. În cazul în care se poate realiza schimbarea datei, server-ul trimite către utilizator un prompt în care îi comunică acest fapt. În caz contrar, utilizatorul primește un prompt care îl informează că data nu este disponibilă, și este întrebat dacă ar dori să introducă altă dată, sau să iasă din competiție.
Server-ul procesează răspunsul și face *UPDATE* bazei de date în funcție de răspunsul oferit de utilizator. Trimite apoi un prompt prin care îl informează pe utilizator despre schimbările efectuate. Utilizatorul se deconectează și apoi închide conexiunea.

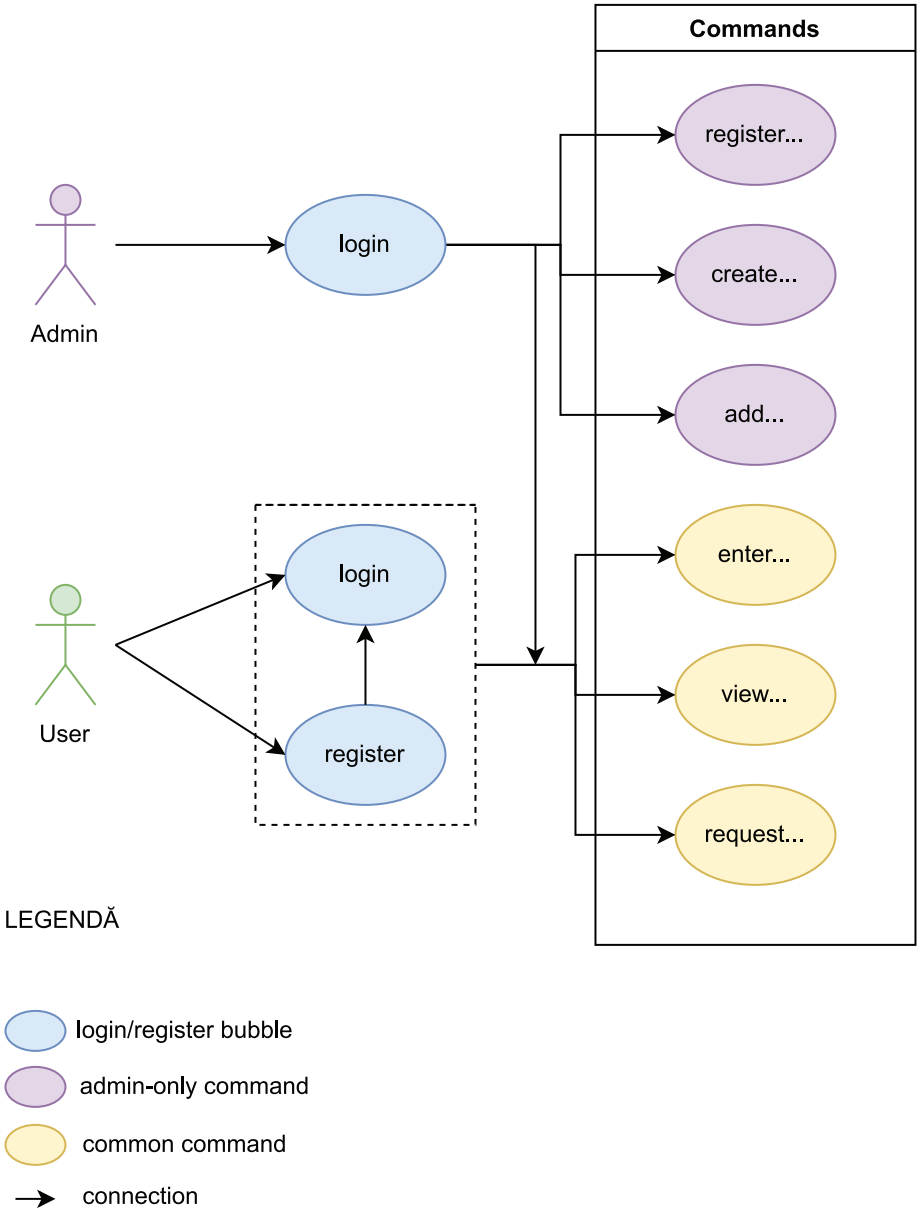

Fig. 3. Use-case diagram

5 Concluzii

Câteva îmbunătățiri care pot fi aduse aplicației sunt:

- realizarea de GUI (interfață grafică)
- posibilitatea unui utilizator de a propune un nou campionat, urmând ca acesta să fie acceptat (sau nu) de către un administrator
- posiblitatea unui utilizator de a-și schimba username-ul
- posiblitatea unui utilizator de a-și schimba parola
- posiblitatea unui utilizator de a-și schimba adresa de contact
- verificarea și permiterea doar anumitor adrese de e-mail (să poată fi acceptate doar *google.com/yahoo.com*)
- utilizarea unei funcții de criptare a parolei utilizatorilor, pentru securitatea informațiilor
- opțiunea de înscriere automată la anumite campionate (ex. un utilizator care se înscrie la o ediție a unui campionat recurent poate alege să fie înscris automat și la edițiile viitoare)

Referințe

1. Transmission Control Protocol, https://ro.wikipedia.org/wiki/Transmission_Control_Protocol
2. Alboaie Lenuța, Programarea în Rețea (III), Curs 6 Rețele de calculatoare (2021)