RemoteJobBoard (B)

Burlica Alexandru, Anul 2, Grupa A1

Facultatea de Informatică, Universitatea A. I. Cuza

1 Introducere

RemoteJobBoard este o aplicație client-server dezvoltată pentru a facilita interacțiunea dintre angajatori și candidați. Aplicația oferă o platformă eficientă pentru procesul de recrutare, permițând utilizatorilor să vizualizeze joburi, să aplice și să le marcheze ca favorite.

Arhitectura aplicației este bazată pe modelul client-server, unde clientul trimite comenzi, iar serverul procesează și răspunde corespunzător. Serverul gestionează conexiuni multiple simultan, asigurând utilizarea eficientă de către mai mulți utilizatori în același timp.

Aplicația include funcționalități de bază, o bază de date care ajută la comunicarea dintre angajat si angajator, bază de date XML ce stochează date precum conturile utilizatorilor, numele job-urilor postate de angajat sau informații despre CV-urile trimise.

2 Tehnologii Aplicate

2.1 Protocol de Comunicație TCP

Aplicația utilizează protocolul TCP pentru a asigura o comunicare fiabilă între client și server. TCP garantează livrarea completă și corectă a datelor, fiind ideal pentru gestionarea comenzilor, precum listarea și aplicarea la joburi.

2.2 Interfața Grafică FLTK

FLTK (Fast Light Toolkit) este folosit pentru a crea o interfață grafică simplă și intuitivă. Aceasta include butoane pentru listarea joburilor, aplicare și marcarea favoritelor, oferind o experiență prietenoasă utilizatorului.

2.3 Multi-threading pe Server

Serverul gestionează conexiuni simultane utilizând thread-uri, asigurând un răspuns rapid pentru fiecare client. Această arhitectură permite scalabilitate și eficiență în procesarea comenzilor.

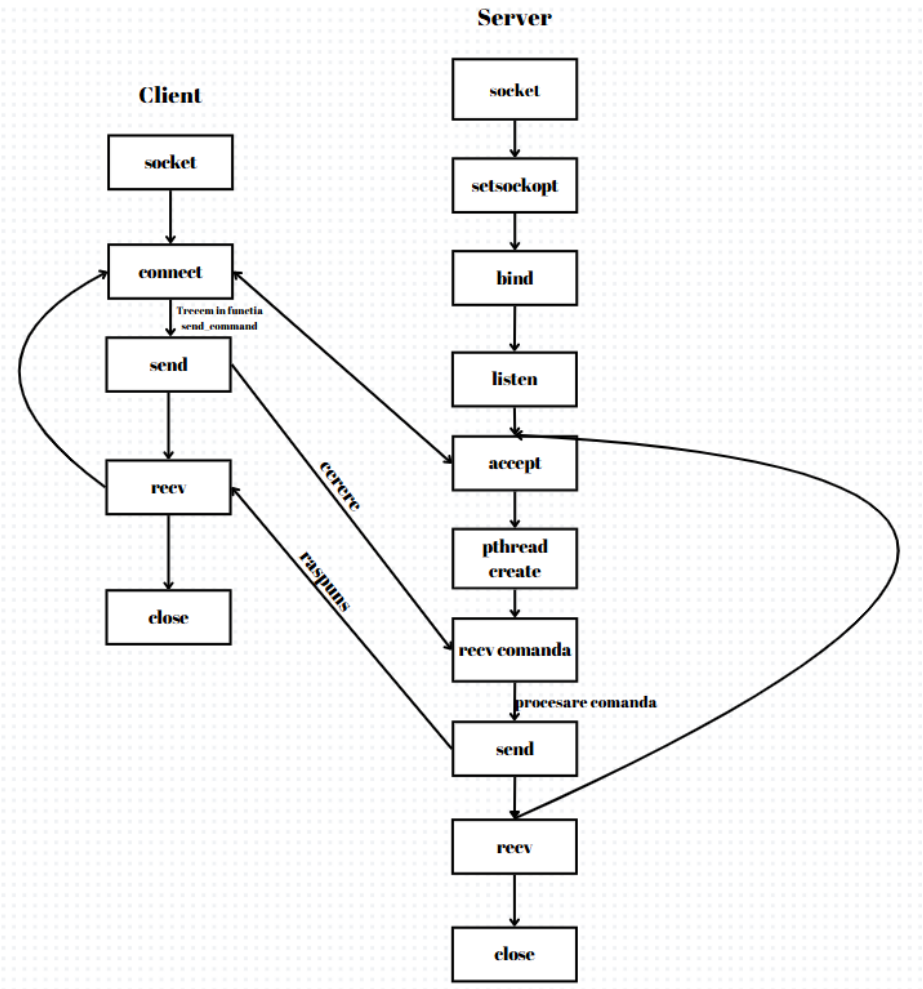
2.4 Bază de date

Aplicația folosește mai multe fișiere de tip XML unde sunt stocate informații cu privire la job-urile postate de angajatori sau CV-urile trimise de către angajați.

3 Structura Aplicației

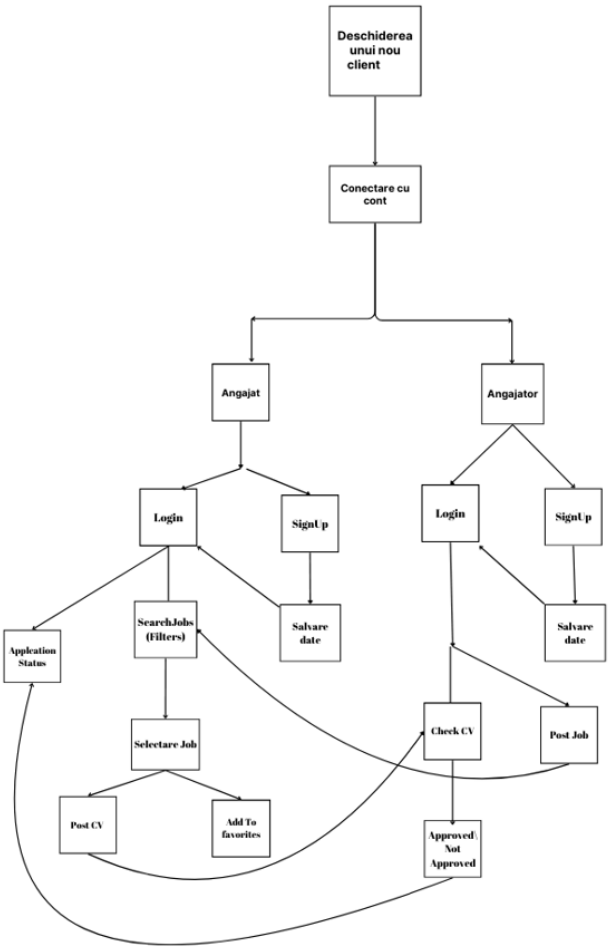
Modelul TCP concurent pe bază de thread-uri este reprezentat mai jos, pe scurt. Clientul se va conecta la server folosind un port dat, urmând să se citească într-o buclă diferite comenzi, până când clientul închide aplicația. Serverul va crea un nou thread pentru fiecare client și va aștepta comenzile acestuia.

3.1 Diagrama TCP



3.2 Diagrama Funcționalităților

Următoarea diagramă ilustrează funcționalitățile, permițând utilizatorilor să își creeze un cont care le va oferi diferite opțiuni ca angajat sau angajator. Fiecare funcție va returna o eroare în cazul în care nu a fost efectuată corect (de exemplu, este utilizat un user care nu a fost niciodată logat). De asemenea, datele privitoare la job-uri, CV-uri sau conturile utilizatorilor sunt stocate în mai multe fișiere de tip XML, baza de date ajutându-ne sa preluăm informațiile mai ușor, având o comunicare fluidă.



4 Aspecte de Implementare

Implementarea aplicației RemoteJobBoard folosește cod eficient pentru gestionarea comunicării client-server, inclusiv utilizarea thread-urilor pentru conexiuni multiple și manipularea comenzilor.

4.1 Comunicarea client-server

Funcția `sendcommand()` este responsabilă pentru transmiterea comenzilor de la client către server și primirea răspunsurilor. Este folosită în interiorul altor funcții, precum cea de logare sau înregistrare, și trimite date despre utilizatori serverului. Aceasta gestionează erorile de conectare și afișează răspunsurile primite:

```
void send_command(const char *command, const char *param1, const char *param2, con
    char buffer[BUFFER_SIZE];
    snprintf(buffer, BUFFER_SIZE, "%s %s %s %s", command, param1, param2, param3);
    send(client_socket, buffer, strlen(buffer), 0);

    memset(buffer, 0, BUFFER_SIZE);
    int n = recv(client_socket, buffer, BUFFER_SIZE, 0);

    if (n <= 0) {
        strcpy(response, "Connection closed by server.");
        close(client_socket);
        exit(0);
    }

    strcpy(response, buffer);
}
```

Fig. 1: Funcția pentru comunicarea client-server

4.2 Funcțiile asociate butoanelor

Aceasta de mai jos este o funcție callback în interiorul unei alte funcții, care facilitează folosirea butonului POSTJOB. Convertește datele la un array de pointeri de tip FLinput, pentru a le stoca într-un buffer și a le trimite serverului, ca mai apoi sa primească un răspuns. Funcția este esențială atunci când apăsăm butonul.

```
submit_button->callback([](Fl_Widget *, void *data) {
    Fl_Input **inputs = (Fl_Input **)data;
    const char *titlu = inputs[0]->value();
    const char *descriere = inputs[1]->value();
    const char *salar = inputs[2]->value();

    if (strlen(titlu) == 0 || strlen(descriere) == 0 || strlen(salar) == 0) {
        response_output->value("All fields must be filled!");
        return;
    }

    char buffer[BUFFER_SIZE];
    snprintf(buffer, BUFFER_SIZE, "POST_JOB %s|%s|%s", titlu, descriere, salar);
    send(client_socket, buffer, strlen(buffer), 0);

    memset(buffer, 0, BUFFER_SIZE);
    recv(client_socket, buffer, BUFFER_SIZE, 0);

    response_output->value(buffer);

}, new Fl_Input *[3]{job_title, job_description, job_salary});
```

Fig. 2: Exemplu de funcție asociată butoanelor

4.3 Interfața Grafică

Folosim FLTK pentru a crea o interfață grafică simplă și plăcută ochiului, care include pentru început o pagină destinată logării sau înregistrării. Mai apoi, în funcție de tipul contului creat, va facilita o pagină pentru angajați sau angajatori cu diferite opțiuni.

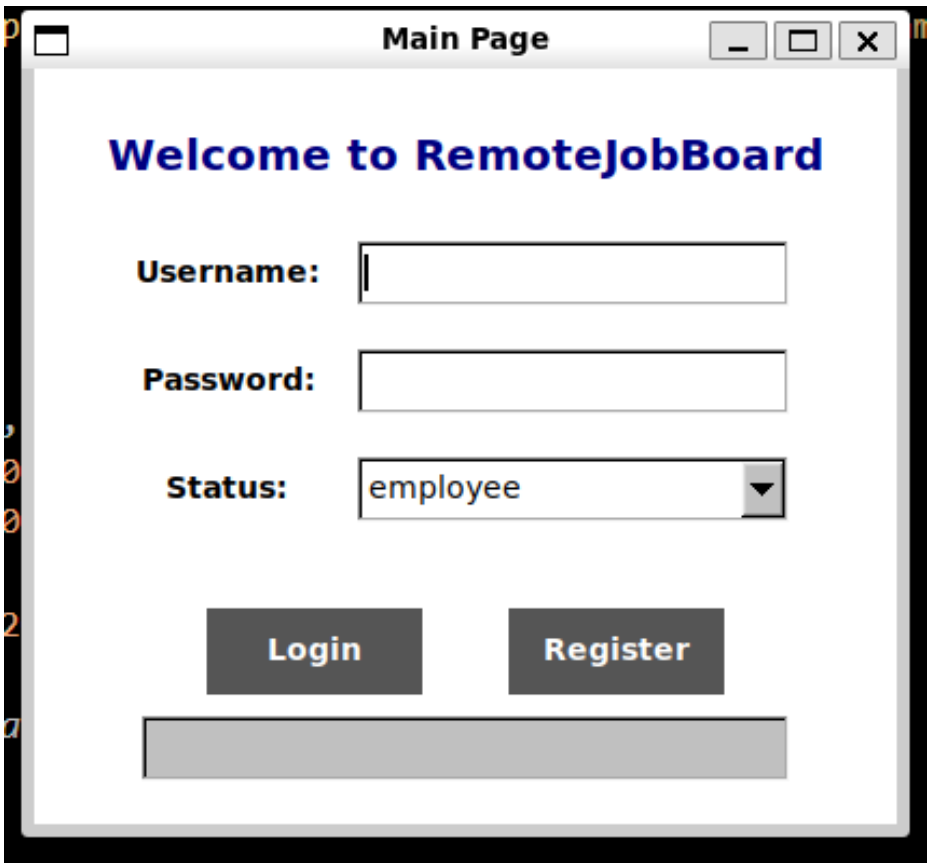
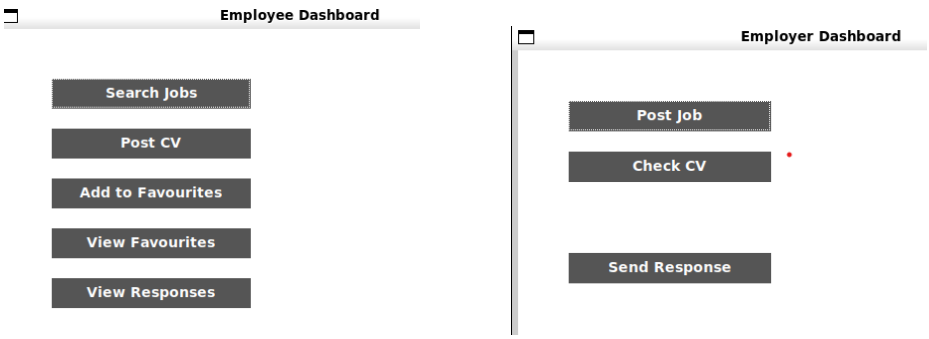


Fig. 3: Interfața grafică



(a) Interfața pentru angajat (b) Interfața pentru angajator

Fig. 4: Interfețe grafice pentru angajat și angajator

4.4 Thread-uri

Serverul gestionează fiecare client printr-un thread dedicat. Funcția `handle_client()` primește comenzi de la client, le procesează (ajutându-se de funcții) și trimite răspunsuri. Funcția este folosită atunci când thread-ul este creat, facilitând astfel implementarea concurenței. Pentru fiecare comandă dispusă de client, se va emite un răspuns bazat pe ce cere.

```
} else if (strcmp(command, "POST_JOB") == 0) {
    char titlu[BUFFER_SIZE], descriere[BUFFER_SIZE], salar[BUFFER_SIZE];
    sscanf(buffer + strlen("POST_JOB") + 1, "%[^|]|%[^|]|%[^\n]", titlu, descriere, salar);
    add_job(titlu, descriere, salar);
    send(client_socket, "Job added successfully!", 24, 0);
} else if (strcmp(command, "SEARCH_JOBS") == 0) {
    char response[BUFFER_SIZE] = {0};
    get_all_jobs(response);
    send(client_socket, response, strlen(response), 0);
} else if (strcmp(command, "POST_CV") == 0) {
    char titlu[BUFFER_SIZE], nume[BUFFER_SIZE], email[BUFFER_SIZE], experienta[BUFFER_SIZE];
    sscanf(buffer + strlen("POST_CV") + 1, "%[^|]|%[^|]|%[^|]|%[^\n]", titlu, nume, email, experienta);
    add_cv(titlu, nume, email, experienta);
    send(client_socket, "CV posted successfully!", 24, 0);
}else if (strcmp(command, "CHECK_CV") == 0) {
    char titlu[BUFFER_SIZE] = {0};
    sscanf(buffer + strlen("CHECK_CV") + 1, "\"%[^\"]\"", titlu); // citire cu tot cu spatii
    char response[BUFFER_SIZE] = {0};
    get_cvs_for_job(titlu, response);
    send(client_socket, response, strlen(response), 0);
```

Fig. 5: Exemplu de comenzi procesate de server

```
    int *client_socket_ptr = malloc(sizeof(int));
    *client_socket_ptr = client_socket;

    pthread_t thread_id;
    if (pthread_create(&thread_id, NULL, handle_client, client_socket_ptr) != 0)
        perror("Thread creation failed");
        close(client_socket);
        free(client_socket_ptr);
    }

    pthread_detach(thread_id);
}
```

Fig. 6: Crearea și gestionarea thread-urilor pentru clienți

4.5 Protocolul Aplicației

Aplicația implementează un protocol simplu la nivel de aplicație, bazat pe schimbul de mesaje între client și server. Protocolul definește următoarele comenzi:

- **SEARCH_JOBS**: Solicită serverului lista joburilor disponibile.
- **POST_CV**: Trimite o cerere de aplicare pentru jobul specificat.
- **FAVOURITE_JOB**: Salvează jobul selectat în lista de favorite.
- **VIEW FAVOURITES**: Prezintă lista de job-uri adăugate la favorite.
- **VIEW RESPONSES**: Vede răspunsurile la cererile de aplicare.
- **POST JOB**: Permite angajatorului să posteze un job nou ce poate fi văzut de potențiali angajați.
- **CHECK CV**: Verifică aplicațiile în funcție de job.
- **SEND RESPONSE**: Trimite răspuns la cererea de aplicare pentru job.

Fiecare comandă este procesată de server și mapată la o funcție specifică. De exemplu: - Comanda `SEARCH JOBS` este mapată la o funcție care returnează lista joburilor stocate în baza de date pentru job-uri

4.6 Scenarii Reale de Utilizare

Pentru a înțelege modul în care aplicația va este utilizată, vom descrie câteva scenarii practice:

Scenariu 1: Vizualizare Joburi și Aplicare 1. Utilizatorul pornește aplicația client și se conectează la server. 2. Apasă pe butonul "SEARCH JOBS", trimițând comanda `SEARCH JOBS`. 3. Serverul trimite lista joburilor, iar clientul le afișează în interfață. 4. Utilizatorul apasă pe POST CV, se trimite comanda, selectează job-ul dorit și completeaza cerințele. 5. Serverul confirmă aplicarea.

Scenariu 2: Înregistrare și Login 1. Utilizatorul completează datele pentru înregistrare, care sunt trimise serverului folosind comanda `REGISTER`. 2. Serverul stochează datele în baza de date. 3. Utilizatorul folosește ulterior comanda `LOGIN` pentru a accesa funcționalitățile aplicației.

5 Concluzii

RemoteJobBoard oferă o platformă simplă, eficientă și extensibilă pentru procesul de recrutare. Utilizatorii pot interacționa cu aplicația pentru a vizualiza joburi, a aplica și a gestiona informații despre angajări. În scenarii reale, un angajat poate utiliza funcționalități precum `SearchJobs`, `AddToFavorites`, și `PostCV`, în timp ce angajatorii pot verifica CV-urile și posta joburi noi.

Integrarea bazei de date permite gestionarea persistentă a datelor despre utilizatori, joburi și CV-uri. Acest lucru facilitează extinderea funcționalităților aplicației, incluzând autentificarea utilizatorilor, navigarea simplă prin lista de job-uri sau CV-uri. Aplicația are potențialul de a deveni un instrument complet pentru recrutare, simplificând procesele atât pentru angajați, cât și pentru angajatori.

6 Referințe Bibliografice

References

1. Wikipedia, "TCP Protocol," `https://en.wikipedia.org/wiki/Transmission_Control_Protocol`.
2. FLTK Documentation, `https://www.fltk.org/documents.php`.
3. Alboaie Lenuța, Panu Andrei, "Rețele de Calculatoare - Facultatea de Informatică," `https://edu.info.uaic.ro/computer-networks/cursullaboratorul.php`.
4. Arpit Bhayani, "Why most TCP servers are multi-threaded and how to build one from scratch," LinkedIn, `https://www.linkedin.com/pulse/why-most-tcp-servers-multi-threaded-how-build-one-from-arpit-bhayani`.