Universitatea Alexandru Ioan Cuza Iași
Facultatea de Informatică

Romila Vlad-Alexandru

An universitar: 2021-2022

Grupa: 2A1

1 Introducere

Proiectul pe care l-am ales se numește "Drag'n'drop processes", având scopul de a ajuta utlizatorul să schematizeze o comandă înlănțuită în linux. Înainte de a trece la partea de schematizare, utilizatorul este rugat să își creeze un cont sau să se autentifice în cazul în care are deja un cont. Odată autentificat, *user*-ul va putea începe o nouă schemă sau să continue una deja existentă, schemă ce poate fi selectată în secțiunea **Istoric**.

Schema unei comenzi înlănțuite este formată din mai multe comenzi simple interconectate. Odată ce utilizatorul apasă pe o comandă din componența schemei, un *modal* ce are scopul de a capta parametrii comenzii selectate va apărea.

2 Tehnologiile Folosite

La bază, aplicația este de tip client/server, iar din acest motiv am ales să implementez un server TCP concurent ce va fi găzduit pe un VPS.

TCP (Transmission Control Protocol) este un protocol de transport orientat conexiune, fara pierdere de informații. Caracteristica cea din urmă este probabil cea mai importantă în cadrul proiectului ales, fiind necesar să salvăm cu exactitate istoricul schemelor unui utilizator, fără a risca salvarea parțială sau coruptă a unei scheme.

Pentru realizarea concurenței între clienți este folosit $fork$-ul, creându-se o zonă de memorie pentru fiecare client conectat.

Pentru a crea o interfață grafică cât mai intuitivă și plăcută, este folosită librăria $SFML$ (Simple and Fast Multimedia Library), ce ușurează implementarea diferitelor ecrane din cadrul aplicației *client*. Pentru distribuirea datelor între server și client dar și pentru stocarea locală a datelor, am folosit structuri JSON folosind librăria *nlohmann/json*. Această librărie permite cu ușurință inițializarea, editarea și transformarea în text a structurilor JSON.

3 Arhitectura proiectului

Pe VPS-ul unde este găzduit serverul rulează și o bază de date MySQL. Serverul este conectat la această baza de date, fiind vitală în administrarea conturilor utilizatorilor și în salvarea schemelor create de aceștia.

După cum se poate vedea și în figura 1, structura bazei de date este foarte simplă. Tabela "*users*" memorează datele fiecărui utilizator al aplicației, iar tabela "*commands*" conține schemele salvate de fiecare utilizator în parte, acestea urmând să fie returnate ca răspuns de către server atunci când utilizatorul vizitează secțiunea **Istoric**. Clientul apelează serverul în următoarele situații:

- La autentificare pentru a verifica datele. Dacă datele sunte corecte, utilizatorul este autentificat;
- La înregistrare pentru a crea o nouă entitate în tabela *users*. Odată ce datele utilizatorului au fost salvate, serverul permite autentificarea;

- Pentru a obține lista schemelor realizate de utilizator. Serverul caută toate schemele din tabela "*commands*" unde atributul *user_id* este același cu *id*-ul utilizatorului;
- Pentru a șterge o schemă salvată anterior. Serverul va șterge schema din baza de date și va returna noua lista de scheme realizate de acel utilizator.
- Pentru a salva o schemă. În cazul in care utilizatorul a editat o schemă deja existentă, serverul va edita schema respectivă. Dacă schema care se dorește a fi salvată nu exista in tabela "*commands*", serverul o va adăuga în tabelă.
- Pentru a redenumi o schema salvată. Serverul va schimba numele schemei selectate și va returna noua listă de scheme salvate.
- Pentru a rula o schema creată. Serverul va rula comanda înlănțtuită si va returna răspunsul.

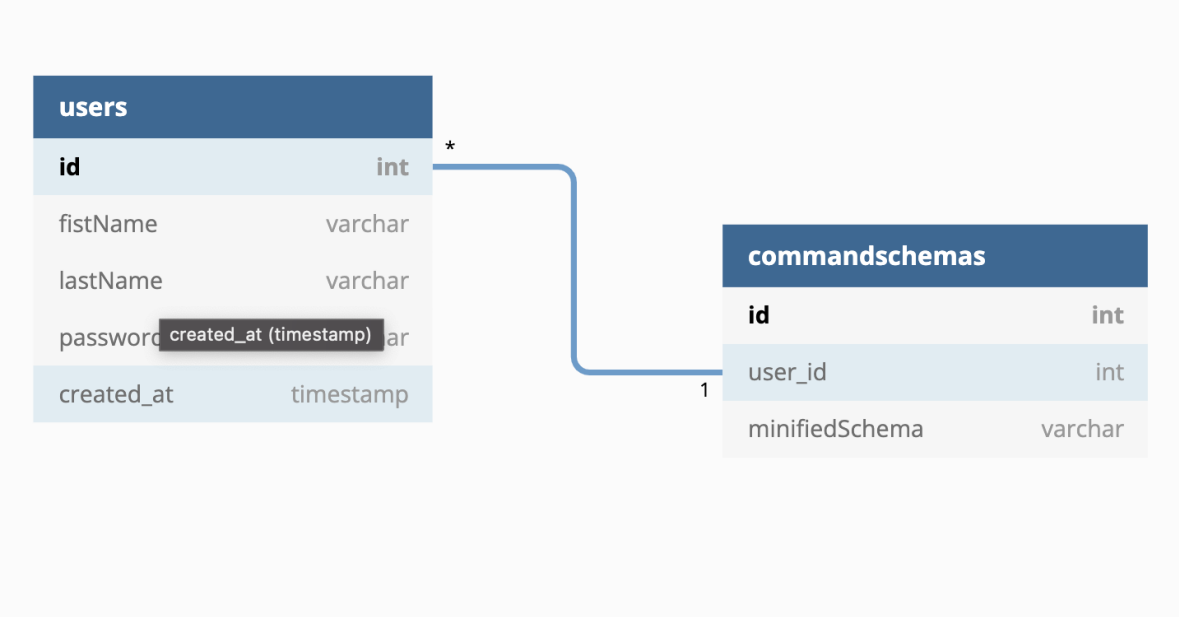

Figure 1: Structura bazei de date

4 Detaill de implementare

Provocarea principală a proiectului este redată de partea grafică. În cadrul aplicației, utilizatorul poate folosi un *playground* în care poate selecta diferite comenzi simple pe care ulterior le va ordona și conecta între ele, formând o schemă ce ușurează vizualizarea unei comenzi inlănțuite complexe. Schema poate fi apoi rulată, serverul procesând comanda înlănțuită si returnând răspunsul. Ce poate fi trecut cu vederea însă este extrem de important este logica din spatele formării din punct de vedere grafic a schemei.

Schema reprezintă în cadrul codului o clasă, *CommandMaker*, ce înglobează toate comenzile simple adăugate, conexiunile dintre ele, și acțiunile de *drag and drop*, *scale*, *rotate* ce sunt ireversibile.

O comandă simplă este reprezentată de clasa *Command*, ce memorează în timp real poziția exactă a componentei în cadrul schemei și are ca și metode:

- "rotate" ce rotește componenta;
- "scale" ce mărește sau micșorează componenta;
- "draw" ce ajută la randarea componentei respective in cadrul *playground*-ului .

Orice componentă care poate fi desenată este formată din mai multe subcomponente: cercuri, pătrate, linii etc. Penru a ușura desenarea acestor subcomponente, o componentă simplă este tratată ca o matrice, iar coordonatele subcomponentelor sunt raportate la originea componentei parinte, nu la originea *playground*-ului.

Cu ajutorul librăriei $SFML$, se pot crea cu ușurință diferitele *ecrane* prin care utilizatorul trece (autentificare, înregistrare etc.). Cu ajutorul acestor ecrane, se poate vizualiza cu ajutorul figurii 2 un $user journey$ generalizat.

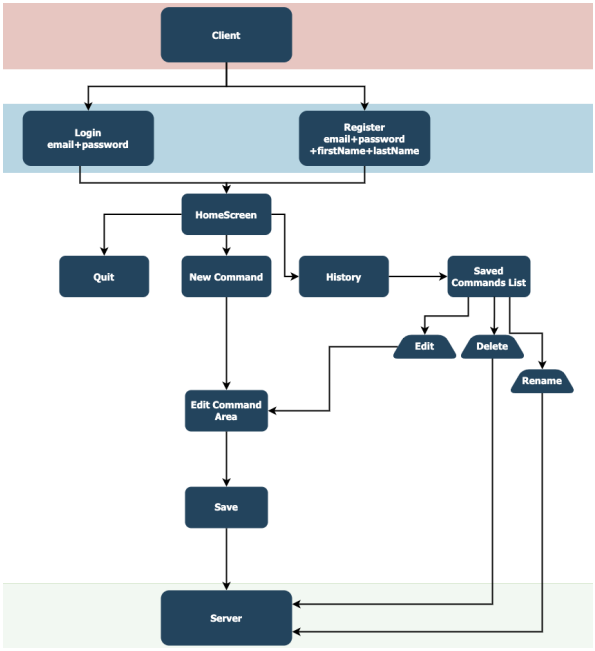

Figure 2: Diagrama

Odată intrat în aplicație, utilizatorul are acces la ecranul de autentificare. Dacă acesta are deja un cont, va trebui să introducă email-ul și parola contului, putând apoi să aiba acces la meniul aplicației.

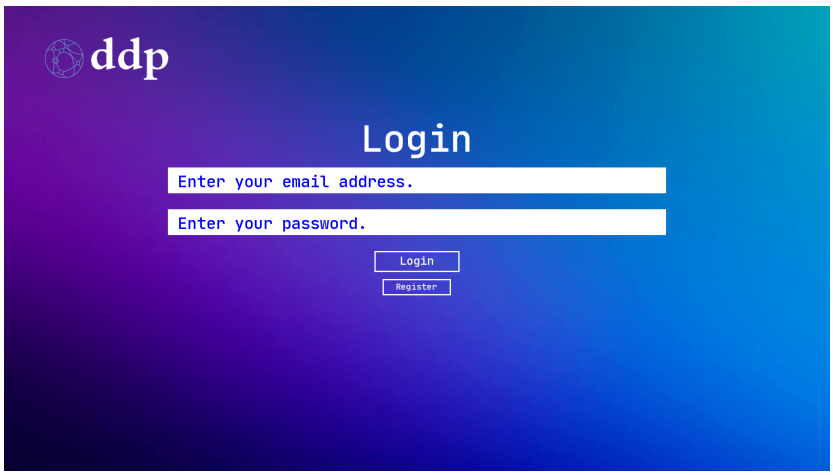

Figure 3: Ecrnanul de autentificare

Meniul aplicației îi permite utilizatorului să deschidă pagina de profil, să inceapă o nouă comandă, să vadă comenzile salvate sau să iasă din aplicație.

Ecranul de creare comenzi conține în partea din stânga a ecranului diferite comenzi simple din suita de comenzi UNIX. Odată ce o comanda este selectată, aceasta poate fi adaugată oriunde pe ecran prin

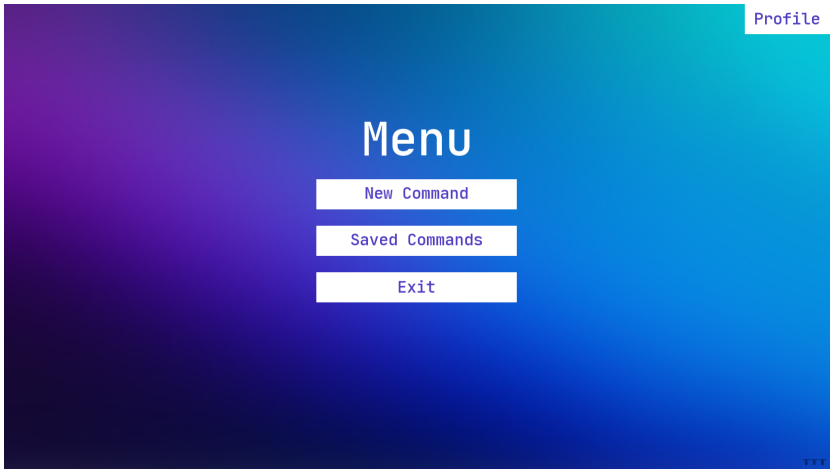

Figure 4: Ecranul meniului

simpla apăsare a spațiului negru. Între două sau mai multe comenzi, se pot adauga linii de legătură, iar odată ce o comanda este completă(are comenzi simple conectate de la stânga spre dreapta), butoanele "*Run*" și "*Runlocally*" vor apărea pe eccran. Cu ajutorul butonului de "*Run*", comanda desenată va fi rulată pe VPS-ul descris mai sus, clientul primind apoi răspunsul din urma rulării. Apăsând pe butonul de "*Runlocally*", comanda desenată va fi rulată local.

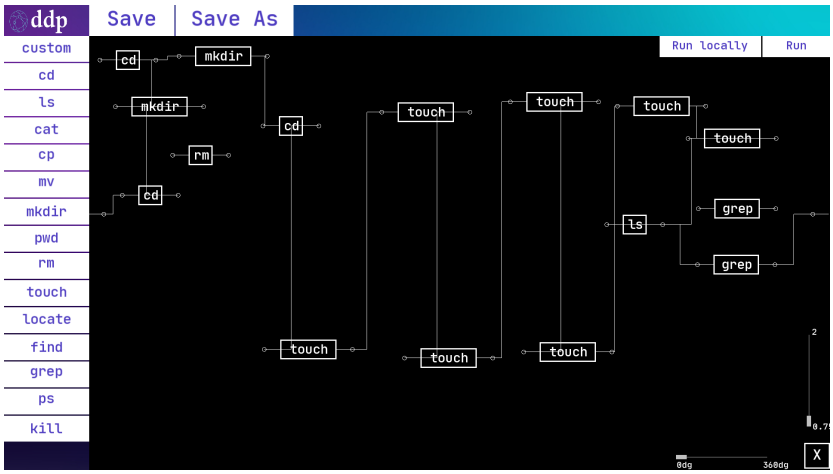

Figure 5: Creare comanda inlantuita

5 Concluzii

În acest moment, aplicația funcționează corespunzător, comenzile și mesajele aferente sunt parsate corect, însă există anumite puncte care pot fi îmbunătățite. În primul rand, utilizatorul ar putea să își seteze el însuși lista de comenzi simple din care poate alege în loc să fie limitat la o listă predefinită. În al doilea rând, ar trebui efectuată funcționalitatea butonului "*custom*" din ecranul de creare comezni, buton prin care utilizatorul poate adăuga comenzi ce nu se găsesc în partea stângă a ecranului.

6 Bibliografie

- https://profs.info.uaic.ro/ computernetworks/cursullaboratorul.php
- https://www.sfml-dev.org/documentation/2.5.1/

- https://www.geeksforgeeks.org/tcp-server-client-implementation-in-c/
- https://www.geeksforgeeks.org/differences-between-tcp-and-udp/
- https://dev.mysql.com/doc/
- https://www.jobsity.com/blog/5-reasons-why-mysql-is-still-the-go-to-database-management-system/
- https://profs.info.uaic.ro/ ioana.bogdan/
- https://stackoverflow.com/
- https://github.com/nlohmann/json