UNIVERSITATEA DIN BUCUREŞTI

FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ

SPECIALIZAREA INFORMATICĂ

Lucrare de licenţă

APLICAŢIE MOBILĂ ÎN PLATFORMA ANDROID PENTRU SOCIALIZARE ŞI VIZIONARE DE VIDEOCLIPURI - CHATOGETHER

Absolvent

Radu George-Marian

Coordonator ştiinţific

Lect. Univ. Dr. Ana Cristina Dăscălescu

Bucureşti, iulie 2024

Cuprins

Introducere

Proiectul meu de licență constă într-o aplicație destinată socializării sigure între utilizatori prin mesaje text criptate end-to-end și vizionării sincronizate de videoclipuri de pe YouTube, fiind dezvoltată pentru platforma Android, versiunea 14. Motivația ce stă în spatele temei alese constă în combinația dintre curiozitatea privind procesul de dezvoltare a unei aplicații mobile, dar și a faptului că nu există, momentan, o aplicație disponibilă publicului larg care să ofere atât funcționalități pentru gestionarea conversațiilor de tip chat, private și de grup, în care mesajele trimise sunt criptate end-to-end, cât și posibilitatea creării de camere pentru vizionare sincronizată de videoclipuri în mod independent de aceste conversații.

Aplicații similare

Pentru sistemul de operare ales există mai multe aplicații ce oferă posibilitatea de socializare între utilizatori prin conversații de tip chat. Un astfel de exemplu este aplicația WhatsApp, deținută de compania Meta. WhatsApp este una dintre cele mai populare platforme de socializare din lume și oferă aplicații pentru mai multe medii de rulare, printre care și o versiune pentru sistemul de operare Android. Similar cu aplicația dezvoltată de către mine, WhatsApp oferă posibilitatea de gestionare a conversațiilor de tip chat private sau de grup și trimiterea de mesaje criptate end-to-end în aceste conversații. Spre deosebire de aplicația cunoscută la nivel mondial, ce se bazează pe utilizarea numerelor de telefon și a listei de contacte din dispozitivele mobile ale utilizatorilor, ChaTogether implementează un sistem de prietenii pentru a permite utilizatorilor să creeze conversații. Astfel, conturile nu necesită un număr de telefon, iar stabilirea relațiilor cu alți utilizatori în cadrul aplicației este independentă de informațiile din lista de contacte a utilizatorului ce o folosește.

În ceea ce privește vizionarea de videoclipuri, una dintre cele mai faimoase aplicații disponibile pentru sistemul de operare Android ce oferă această funcționalitate este numită Rave – Watch Party. Folosind această aplicație, utilizatorii pot crea camere de vizionare a videoclipurilor de pe mai multe platforme în mod sincronizat, inclusiv de pe YouTube. Spre deosebire de aceasta în care există camere publice și private, dar și un chat în fiecare cameră, aplicația ChaTogether implementează în mod independent conceptul de conversație, iar camerele de vizionare a videoclipurilor sunt private, putând fi accesate doar de către acei utilizatori ce dețin codul unic de conectare. De asemenea, conversațiile sunt implementate în

mod independent pentru a asigura persistența mesajelor și o separare clară a funcționalităților.

Prezentarea capitolelor lucrării

În cadrul acestei lucrări sunt descrise mai multe aspecte ce privesc aplicația dezvoltată, organizate într-un mod clar, pe mai multe capitole. Aceste capitole sunt denumite, în această ordine: **Tehnologii utilizate**, **Algoritmi criptografici**, **Arhitectura aplicației, Funcționalitățile aplicației** și **Testarea aplicației**. Fiecare capitol este descris pe scurt în paragrafele următoare.

În capitolul **Tehnologii utilizate** sunt prezentate limbajele de programare, serviciile și protocoalele utilizate în procesul de dezvoltare al aplicației. Aici se pot găsi informații generale despre fiecare tehnologie prezentată, de unde se poate deduce motivul alegerii sale.

Următorul capitol, **Algoritmi criptografici** descriu toate procedurile criptografice utilizate pentru asigurarea comunicării sigure între utilizatori. Acesta include informații despre modul de lucru și scopul algoritmului de criptare asimetrică RSA, a celui de criptare simetrică AES, dar și a funcției pentru derivarea de chei criptografice de lungime variată pornind de la parolele utilizatorilor, și anume PBKDF2.

Capitolul **Arhitectura aplicației** surprinde strategiile alese pentru implementarea aplicației. Acesta include prezentarea fiecărei componente din abordarea MVC adoptată, a rolurilor pe care utilizatorii aplicației le pot avea, dar și a modului de gestionare a cheilor criptografice și a criptării, respectiv decriptării, mesajelor din conversații.

Funcționalitățile aplicației este capitolul în care sunt prezentate în amănunt toate acțiunile pe care utilizatorii le pot realiza în aplicația dezvoltată. Sunt incluse și o serie de capturi de ecran pentru a surprinde aspectul aplicației, dar și pentru a descrie într-un mod clar cum poate fi accesată fiecare funcționalitate și cum este afișat rezultatul fiecărei operații către utilizatorul final.

Testarea aplicației reprezintă numele capitolului final al lucrării, ce descrie metodele abordate pentru a testa aplicația dezvoltată, în scopul validării corectitudinii codului scris. Aici sunt prezentate configurările necesare înaintea scrierii suitei de teste, dar apar listate și funcțiile pe care testele scrise le acoperă.

Capitolul 1. Tehnologii utilizate

1.1 Java

Java reprezintă o platformă destinată dezvoltării de aplicații software, ce a fost pentru prima dată publicată de către Sun Microsystems la mijlocul anilor 1990. Principalul scop pe care inventatorii acesteia și l-au propus a fost acela de a elimina necesitatea compilării codului sursă în fișiere executabile specifice mediilor diferite în care acestea aveau să fie rulate. Pentru a reuși acest lucru ei au dezvoltat o mașină virtuală denumită Java Virtual Machine (JVM), ce este în esență un program specializat pentru fiecare sistem de operare, necesar pentru executarea codului Java scris de către programatori [4].

Stabilitatea și fiabilitatea pe care această platformă de dezvoltare le oferă au condus la utilizarea acestui limbaj în numeroase domenii din industria IT. De la implementarea unei simple aplicații web, mobile sau desktop, la crearea de jocuri video, dar chiar și a unor biblioteci și aplicații necesare atât în domeniul geneticii, cât și în cel al ingineriei spațiale, Java s-a dovedit a fi o unealtă de încredere și eficientă ce poate fi folosită în diverse arii de activitate [2].

Dintre aplicațiile construite în platforma Java, unele sunt cu adevărat complexe și remarcabile. Minecraft, un joc video dezvoltat inițial de către Markus Alexej Persson folosind Java, este unul dintre cele mai cunoscute jocuri video la nivel mondial, cu un număr aproximativ de 170 milioane de jucători activi lunar [20]. Unele programe folosite de către cercetătorii de la NASA au fost, de asemenea dezvoltate cu ajutorul Java, printre care se numără: Maestro Mars Rover, programul de control al Spirit Mars Exploration Rover, ce a fost utilizat pentru a ghida acest robot în cadrul explorării planetei Marte și JavaFX Deep Space Trajectory Explorer, un program utilizat pentru calcularea traiectoriilor obiectelor în spațiu [2]. Platforma Java a fost utilizată și în dezvoltarea unor aplicații și biblioteci folosite în domeniul bioinformaticii, precum Integrated Genome Browser, o aplicație folosită pentru a analiza de date referitoare la genomul uman [10], dar și proiectul BioJava, ce oferă suport pentru procesarea datelor biologice prin bibliotecile pe care le pune la dispoziție [19].

1.1.1 Limbajul de programare Java

Limbajul de programare Java poate fi descris succint ca fiind un limbaj de programare de nivel înalt, ce abordează o paradigmă de programare orientată pe obiecte, cu o sintaxă similară a limbajele C sau C++.

Limbajul este de asemenea caracterizat ca având variabile cu tip static (statically-typed), ceea ce înseamnă că verificarea tipurilor de date se face în momentul compilării, acolo unde acest lucru este posibil, fapt ce reduce riscul apariției unei erori la rulare datorată neconcordanței tipurilor de date ale variabilelor. Pentru variabile trebuie declarat, în general, tipul de date pe care aceasta îl va avea, dar este posibilă și inferența automată a tipului de date pentru acestea.

De-a lungul anilor au fost publicate mai multe versiuni ale limbajului, în care au fost adăugate îmbunătățiri. Printre cele mai notabile versiuni ale limbajului se numără Java 5, Java 8, Java 11, Java 21, iar cea mai recentă versiune a limbajului este Java 22. Dintre toate versiunile menționate anterior, probabil cea mai populară a fost Java 8, în cadrul căreia au fost introduse în limbaj lambda expresiile și Stream API, adoptând astfel elemente specifice paradigmei de programare funcțională [14].

1.1.2 Java Virtual Machine (JVM)

Java Virtual Machine reprezintă un program, specific fiecărei arhitecturi hardware și sistem de operare, ce se ocupă de crearea mediului necesar pentru executarea programelor Java dezvoltate de către utilizatori [4].

Pentru ca un program scris in limbajul de programare Java să poată rula pe un calculator, acesta trebuie mai întâi să fie compilat în bytecode. Mai departe, acest bytecode este folosit de către JVM pentru a transmite comenzi sistemului de operare, indiferent pe ce calculator este executat. Această abordare a condus la portabilitatea limbajului, programele Java putând fi executate oriunde există instalat JVM.

O altă caracteristică importantă a Java Virtual Machine este Garbage Collector-ul cu care acesta este echipat. Garbage Collector-ul este un proces ce are rolul de a ține evidența zonelor de memorie pe care programul Java ce rulează le folosește, iar când acesta observă că o anumită zonă de memorie nu mai este utilizată o eliberează automat [13]. Astfel sunt evitate situațiile în care programatorul a omis să elibereze o zonă de memorie, ce reprezintă o problemă majoră în alte limbaje în care alocarea si eliberarea memoriei sunt gestionate manual, precum C sau C++.

1.2 Spring & Spring Boot

1.2.1 Framework-ul Spring

Spring este numele unuia dintre cele mai populare framework-uri construite pornind de la limbajul de programare Java, ce are drept scop facilitarea procesului de dezvoltare a aplicațiilor software. Acesta este un framework gratuit si open-source, ceea ce înseamnă că oricine poate folosi, vedea și chiar schimba codul sursă al framework-ului după cum dorește [15].

Principalele caracteristici pe care acest framework le prezintă sunt beans, dependency injection, stilul modular în care este construit, abordarea paradigmei MVC și gestionarea tranzacțiilor într-un mod declarativ [15].

Termenul de bean se referă la obiectele ce compun aplicația dezvoltată și care sunt gestionate automat de către container-ul IoC (Inversion of Control). Dependency injection reprezintă o metodă de a implementa conceptul de inversiune a controlului (Inversion of Control), și constă în furnizarea de obiecte de către nucleul framework-ului către alte obiecte prin constructor sau metode de tip setter. În acest mod nu mai este necesar ca programatorul să creeze efectiv obiecte de care are nevoie în clasele pe care le construiește, ceea ce duce la scrierea de cod mai curat și ușurință de reutilizare, refactorizare și testare a codului [15].

Stilul modular abordat de acest framework este justificat prin posibilitatea oferită programatorului de a folosi doar modulele de care acesta are cu adevărat nevoie, acestea fiind independente unul față de celălalt, iar fiecare oferind câte o funcționalitate specifică [15].

MVC (Model-View-Controller) reprezintă o modalitate de separare a sarcinilor pe care o aplicație trebuie să le îndeplinească, pentru ca acestea să poată fi tratate independent de către programator și pentru a fi mai ușor de întreținut. Spring oferă în modulul Web-Servlet o implementare proprie a acestei abordări, integrată cu alte funcționalități ale framework-ului [15].

Framework-ul Spring implementează gestiunea tranzacțiilor cu bazele de date într-o manieră declarativă, ușurând astfel munca programatorilor ce vor să includă în aplicațiile lor interacțiunea cu baze de date. Această implementare este compatibilă cu API-uri populare precum Hibernate sau JPA [15].

1.2.2 Spring Boot

Fiind o extensie a framewor-ului Spring, Spring Boot facilitează procesul de configurare al acestui framework, oferind un set predefinit de dependențe uzuale ce pot fi direct folosite de către programator. Cu toate acestea, dezvoltatorii au acces deplin asupra dependențelor ce sunt incluse în proiect și le pot modifica după bunul plac pe măsură ce aplicația este implementată, iar configurația inițială nu se mai potrivește nevoilor acesteia [16].

1.3 Dart

Dart este un limbaj de programare construit pentru a oferi o experiență cât mai plăcută dezvoltatorilor de software. Acesta a fost creat de către compania Google și este la bază un limbaj de programare orientat pe obiecte, care însă integrează și elemente specifice programării funcționale, dar și celei imperative. Este în principiu un limbaj de programare cu scop general, adică ar putea fi folosit pentru numeroase arii de dezvoltare, însă unde acesta excelează este la dezvoltarea front-end-ului aplicațiilor software [17], [7].

Dezvoltatorii limbajului au fost concentrați asupra experienței programatorului, iar din acest motiv au ales ca limbajul Dart să fie cât mai plăcut de utilizat. Din acest motiv s-a ales utilizarea unei sintaxe similare cu cea a limbajelor C și C++, pentru ca trecerea la acest limbaj să fie una cât mai lină. Un alt aspect ce evidențiază importanța experienței programatorilor este alegerea implementării unui sistem de tipuri cu verificare statică, ceea ce înseamnă că variabilele trebuie să aibă asociate un tip de date la compilare, însă nu este necesar ca programatorul să specifice de fiecare dată tipul variabilelor, deoarece acesta poate fi dedus. Cu toate acestea, oferă siguranță în ceea ce privește valorile nule. Orice tip de date, inclusiv cele definite de către programatori cu ajutorul claselor, nu acceptă valori nule a priori, ci trebuie specificat în mod clar folosind simbolul ? că sunt acceptate și valori nule [17].

Un alt aspect caracteristic al acestui limbaj de programare constă în posibilitatea dezvoltării de aplicații ce rulează pe mai multe platforme, precum aplicații web, mobile sau chiar pentru desktop [17].

Deși este un limbaj ce a apărut relativ recent, a dobândit o popularitate extraordinară și prezintă un potențial uriaș datorită framework-ului Flutter.

1.4 Flutter

Având la bază limbajul de programare Dart, Flutter este un framework destinat dezvoltării front-end-ului pentru aplicații software pentru care codul sursă este unic, dar pot rula pe diferite platforme, cum ar fi web, telefoane mobile ce folosesc sistem de operare Android sau iOS, tablete și diverse sisteme de tip desktop, fiind compatibil la momentul actual cu Windows, Linux, dar și macOS. Framework-ul a fost dezvoltat tot de către Google, drept un proiect open-source, și a fost gândit în așa fel încât să facă implementarea aplicațiilor cât mai accesibilă, astfel încât chiar și persoanele cu puțină experiență în acest domeniu să poată învăța rapid cum să o facă [6], [22].

Flutter nu se remarcă doar prin simplitatea codului specifică limbajului de programare Dart, dar aduce numeroase funcționalități menite să faciliteze dezvoltarea de aplicații robuste și eficiente într-un timp cât mai scurt posibil, dintre care cele mai populare sunt timpul scurt de compilare si capabilitatea de hot-reload, numeroasele pachete ce pot fi instalate și documentația bogată oferită de echipa ce dezvoltă framework-ul.

Comparativ cu alte tehnologii ce sunt folosite pentru construirea aplicațiilor software pe mai multe platforme, Flutter prezintă un timp scurt de compilare, iar posibilitatea de a vizualiza modificările făcute în timp real se dovedește a fi foarte folositoare. În plus, codul Dart este compilat în cod ce poate rula direct pe platforma necesară, fără a fi necesară o punte, cum este în cazul React Native, unde se folosește această punte pentru a se comunica între aplicație și platformă, fapt ce rezultă în ineficiență la rulare [6].

Un aspect ce evidențiază ușurința dezvoltării aplicațiilor software în Flutter constă în mulțimea de pachete pe care programatorii le pot instala și care pot fi folosite după o configurare minimală sau chiar fără vreo configurare anterioară. De la simple pachete ce oferă posibilitatea de afișare de mesaje text de tip toast sau de a include noi fonturi în aplicație și până la pachete ce se integrează cu sisteme precum Firebase sau pachete ce oferă comunicare în timp real, cum ar fi Agora, programatorii au la dispoziție pachete cu funcționalități variate și extrem de utile în unele situații.

Documentația unei tehnologii software este foarte importantă pentru programatorii care o folosesc, iar documentația Flutter este una foarte bogată. Pe lângă documentația disponibilă pe website-ul oficial Flutter [22], ce este în sine una cuprinzătoare și ușor de înțeles, programatorii pot instala extensii în mediile lor de dezvoltare software (ex. Visual Studio Code) și pot beneficia de explicații clare despre clasele și metodele folosite prin trecerea peste acestea folosind cursorul mouse-ului. Mai mult decât atât, din documentația oficială sunt disponibile

link-uri către unele videoclipuri explicative, în care sunt prezentate atât elemente incluse în framework, cât și funcționalități oferite de pachete ce necesită instalare.

1.5 MySQL

Bazele de date relaționale sunt cele mai populare tipuri de baze de date, în care informațiile sunt organizate sub formă de tabele între care se formează legături. Acestea stochează datele în niște fișiere existente pe hard-disk, fișiere ce sunt optimizate pentru a suporta interogări complexe de accesare a datelor. Printre cele mai populare exemple de baze de date relaționale existente în industria curentă se numără și MySQL, o bază de date de tip open-source, ce a evoluat de-a lungul timpului în strânsă legătură cu cerințele utilizatorilor, fiind astfel compatibilă cu numeroase alte tehnologii folosite [21].

Popularitatea MySQL se datorează numeroaselor beneficii pe care acest produs le oferă dezvoltatorilor. Fiabilitatea acestei tehnologii software este demonstrată prin numeroasele teste aplicate asupra sa, iar cei 25 de ani de utilizare în cadrul unor aplicații foarte complexe construite de către companii uriașe precum Meta, Uber, Netflix sau Twitter nu au făcut altceva decât să sublinieze acest aspect. De asemenea, echilibrul între performanța ridicată și securitate de nivel înalt sunt caracteristici cheie de care MySQL dă dovadă, alături de ușurința de utilizare și puterea de creștere a bazei de date în funcție de cerințele din ce în ce mai complexe ce apar în cadrul dezvoltării unei aplicații software [21].

Printre cele mai importante exemple de produse ce integrează baze de date MySQL se numără platformele de socializare, precum am menționat anterior, aplicațiile destinate comerțului electronic, dar și aplicațiile pentru gestiunea unor volume mari de conținut, cât și soluții software folosite în cadrul diverselor întreprinderi. În plus, prin intermediul serviciului MySQL HeatWave sunt oferite diverse funcționalități folositoare pentru aplicațiile ce rulează în cloud [21].

1.6 MongoDB

Spre deosebire de bazele de date relaționale, bazele de date NoSQL adoptă alte strategii pentru structurarea datelor pe care le conțin. O astfel de strategie constă în stocarea datelor sub formă de documente JSON. Una dintre cele mai cunoscute baze de date ce folosește documente pentru organizarea informațiilor este MongoDB, ce își propune să facă dezvoltarea de aplicații

cât mai rapidă, oferind baze de date ce pot crește cu ușurință, dar și un mod de backup automat în caz de eșec [8].

Modul de structurare al datelor în documente și colecții ale acestora este unul ce oferă o mai bună înțelegere a informațiilor existente, acestea fiind prezentate într-un format ușor de vizualizat și standardizat în domeniul IT. Mai mult decât atât, este eliminată necesitatea normalizării bazei de date pentru ca organizarea acesteia să fie cât mai eficientă, scutind dezvoltatorii de o muncă în plus. În plus, pentru o eficiență mai mare în momentul interogărilor de tip căutare de informații, MongoDB beneficiază de niște elemente speciale denumite indecși, folosiți pentru a optimiza cât mai mult posibil aceste operații [8].

Duplicarea datelor și stocarea acestora pe mai multe servere este o abordare pe care MongoDB o preia și o îmbunătățește prin sistemul automat de backup ce asigură accesibilitatea datelor în cazul unei eventuale întreruperi a server-ului principal [8]. Acest aspect face ca MongoDB să fie considerată o bază de date de încredere, iar în combinație cu eficiența și structura sa intuitivă, a condus la popularizarea acesteia și la includerea sa în numeroase stive de tehnologii.

1.7 Docker

Docker reprezintă o unealtă pentru rularea de procese în medii izolate față de mediul calculatorului gazdă. Spre deosebire de o mașină virtuală clasică, ce trebuie să ruleze un întreg sistem de operare pe care pot rula diverse aplicații, un container Docker se conectează la motorul Docker, ce este instalat pe calculatorul gazdă, și este suficient să ruleze aplicația pe care o conține fără a mai fi nevoie instalarea unui nou sistem de operare [18].

Imaginile Docker reprezintă un șablon pentru construirea de containere Docker [18]. Prin intermediul acestora poate fi distribuită configurația necesară pentru a crea containere ce conțin anumite elemente. Imaginile Docker pot fi accesate de către utilizatori de pe registrele Docker, cel mai popular exemplu fiind Docker Hub [18]. Folosind această platformă, dezvoltatorii de software pot descărca imagini deja existente și pot de asemenea publica imagini personalizate pentru ceilalți membrii ai comunității Docker.

Docker oferă posibilitatea persistării datelor între rularea containerelor prin intermediul volumelor. Acestea sunt niște obiecte stocate pe hard disk-ul calculatorului gazdă, în care se salvează date necesare aplicației, ce sunt asociate unei anumite zone din sistemul de fișiere al container-elor. În acest mod aplicația ce rulează în container poate accesa fișierele respective ca și cum ar face parte din memoria proprie, dar evident cu beneficiul persistenței acestor date

între rulări succesive.

Principala utilitate a acestui produs software constă în capabilitatea replicării mediului necesar pentru rularea unei aplicații, independent de sistemul de operare pe care aceasta urmează să fie executată. Astfel, un grup de dezvoltatori ce lucrează la aceeași aplicație pot partaja cu ușurință configurația containerelor Docker ce conțin codul respectivei aplicații, pe care le pot rula pe calculatorul propriu fără grija unei eventuale erori de incompatibilitate. În plus, rularea aplicațiilor în containere aduce de la sine beneficii în ceea ce privește securitatea sistemului informatic. Aplicațiile rulează într-un mediu izolat, separat de sistemul principal, astfel fiind limitat accesul aplicațiilor la sistemul de fișiere al computerului gazdă, iar în cazul unei defecțiuni la nivelul aplicației, doar container-ul este afectat, în timp ce calculatorul gazdă nu suferă în urma acesteia.

1.8 Websocket

Websocket reprezintă un protocol ce stabilește modul în care două calculatoare comunică în timp real prin internet. Spre deosebire de protocolul HTTP, unde clienții fac cereri către server, pe care acesta din urmă le procesează și răspunde adecvat, în cadrul protocolului Websocket se stabilește o conexiune bidirecțională și persistentă între ambele mașini ce comunică. Astfel se obține un canal de comunicare, orice computer conectat la el fiind capabil să trimită mesaje, iar cel ce se află la celălalt capăt va primi mesajul cu latență cât mai mică [11].

În figura 1.1 sunt surprinse etapele din cadrul unei conexiuni de tip websocket. În primă instanță aplicația client realizează un apel HTTP către server pentru a cere inițierea conexiunii. Server-ul procesează datele cererii, inclusiv eventuale informații privind autentificarea utilizatorului, și îi răspunde afirmativ, deschizându-se astfel conexiunea bidirecțională. De aici, oricare din cele două părți pot trimite mesaje la orice moment, celălalt computer primind mesajele aproape instantaneu. Conexiunea este încheiată în momentul în care unul dintre cele două calculatoare semnalizează dorința de închidere a canalului de comunicare.

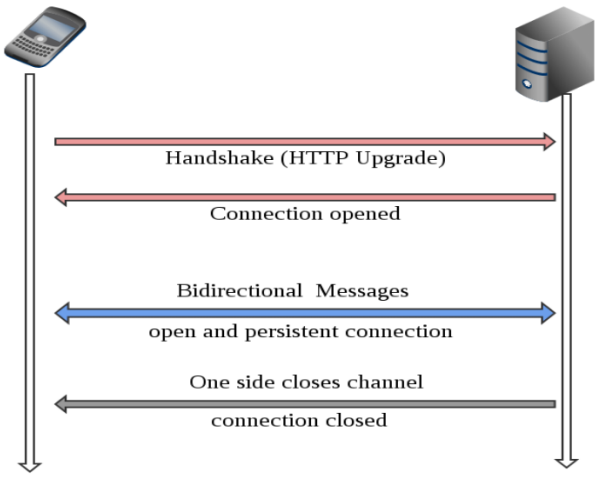

Figura 1.1 Conexiunea Websocket, preluat din [12]

Capitolul 2. Algoritmi criptografici

Aplicația folosește o serie de algoritmi criptografici ce sunt prezentați succint în acest capitol.

2.1 Algoritmul RSA

Probabil cel mai popular algoritm de criptare asimetrica, RSA, ce își primește numele de la inventatorii săi Ron Rivest, Adi Shamir și Leonard Adleman, a fost publicat în anul 1978 pentru a rezolva problema schimbului de chei necesar la criptarea simetrică. Algoritmul prezintă două cazuri principale de utilizare în cadrul erei tehnologizate în care trăim, și anume transmiterea de mesaje ascunse pe care doar cel căruia îi este adresat să îl poată citi, dar și crearea de semnături digitale pentru asigurarea veridicității anumitor mesaje sau fișiere [5].

Acest algoritm are un adevărat fundament matematic în teoria numerelor. Modul în care versiunea de bază a acestui algoritm funcționează este următorul [5]:

1. Etapa de generare a modului RSA:

   În această etapă sunt generate două numere prime mari, ce au anumite proprietăți matematice speciale, care se numesc de regulă p și q. După acest pas, va fi calculat numărul n ca produs a celor două numere prime, și, totodată, se va calcula și φ(n), unde φ se numește indicatorul lui Euler și reprezintă numărul de numere naturale pozitive mai mici decât n și coprime cu acesta. În acest caz particular vom avea

$$\varphi(\mathrm{n}) = (p - 1) * (q - 1)$$

   întrucât ambele numere, p și q, sunt prime, deci φ(p) = p – 1 și φ(q) = q – 1 , iar cum n = p * q, va rezulta egalitatea de mai sus.

2. Etapa de generare a cheilor de criptare si de decriptare:

   Se vor genera în cadrul acestei etape două numere, e și d, după următoarele reguli:

   - Numărul d va fi un întreg mare din intervalul [0, φ(n)), cu proprietatea că numerele d și φ(n) sunt coprime.

- Numărul e va fi tot un număr întreg ales din intervalul [0, φ(n)), care respectă următoarea proprietate:

$$e * d = 1 \left(mod\ \varphi(\mathrm{n})\right)$$

După calculul acestor două numere, cheia de criptare va fi perechea (e, n), iar cheia de decriptare va fi formată din perechea (d, n).

3. Algoritmul de criptare:

Obținerea mesajului criptat c din mesajul clar m, unde m și c sunt considerate reprezentări numerice ale unor mesaje de transmis pe rețea, se face în felul următor:

$$c = m^e (mod\ n)$$

4. Algoritmul de decriptare:

Pentru a decripta mesajul c primit și a obține înapoi mesajul m' care este identic cu mesajul m trimis inițial se procedează în felul următor:

$$m' = c^d (mod\ n)$$

Securitatea algoritmului RSA se bazează pe problema factorizării modulului RSA care spune că un adversar cu putere de calcul probabilist polinomială va reuși să factorizeze numărul n și să găsească cele două numere prime p și q cu o probabilitate neglijabilă [5].

Este, însă necesar de menționat că această variantă poate fi supusă la numeroase atacuri, de exemplu cel prin care un individ rău intenționat poate modifica cu ușurință conținutul mesajului transmis, fără a-l putea vedea, printr-o simplă înmulțire a două mesaje, deja criptate cu aceeași cheie publică. Pentru acest motiv se folosesc atât semnături digitale, cât și alte elemente criptografice pentru a se putea verifica originea mesajului și pentru a asigura faptul că acesta nu a fost alterat.

2.2 Algoritmul AES

Algoritmul Advanced Encryption Standard (AES) este un algoritm de criptare simetrică publicat de către NIST în anul 2000. Acesta a apărut în contextul descoperirii unor slăbiciuni ale algoritmului DES (Data Encryption Standard), și a fost gândit drept un înlocuitor pentru acesta [1].

Spre deosebire de sistemele de criptare asimetrică, în care există o cheie de criptare și

una de decriptare pentru fiecare utilizator, în cazul sistemelor de criptare simetrică, inclusiv AES, cheia de criptare reprezintă în același timp și cheia de decriptare. Din această cauză, este de la sine înțeles că pentru a se realiza comunicarea criptată între două părți, folosind algoritmul AES, este necesar ca ambii participanți la conversație să dețină aceeași cheie unică. Acest lucru implică nevoia unei strategii pentru distribuirea într-un mod securizat a cheii simetrice, ceea ce reprezintă principalul dezavantaj al acestui tip de sistem criptografic.

Avantajele pe care această strategie de criptare le prezintă sunt, însă, destul de semnificative, mai ales în ceea ce privește eficiența și securitatea sa. Datorită design-ului său, algoritmul AES prezintă o performanță foarte bună, comparativ cu algoritmii de criptare simetrică precedenți, inclusiv DES. Eficiența sa a făcut posibilă implementarea acestuia direct în componente hardware și utilizarea sa în noduri de comunicare din rețelele de calculatoare. În plus, după o testare riguroasă s-a dovedit că acest algoritm prezintă o securitate ridicată pe lângă predecesorul său, DES, nepermițând atacatorilor să divulge mesajul criptat [1].

Modul în care algoritmul AES funcționează pentru criptare și decriptare se poate vedea în figura 2.1

Se poate observa cum atât rutina de criptare, cât și cea de decriptare sunt alcătuite din mai multe runde prin care mesajul inițial, fie că este mesaj clar sau criptat, este trecut. De asemenea, fiecare rundă, ce constă în mai multe operații efectuate asupra mesajului, primește o anumită porțiune din cheia secretă, care trece la rândul ei printr-un proces de expansiune.

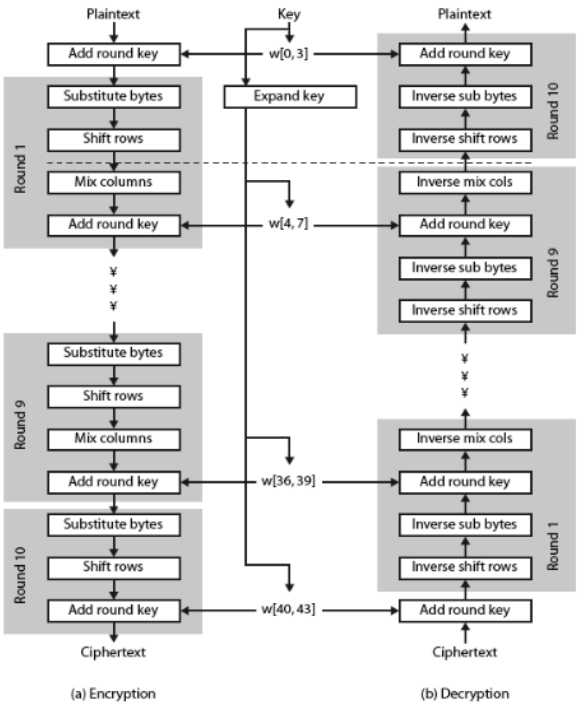

Figura 2.1 Criptare și decriptare folosind AES, preluat din [1]

Algoritmul AES, în forma ilustrată în figura de alături poate cripta doar blocuri de mesaje ce au o dimensiune de exact 128 biți [1]. Pentru criptarea unor mesaje de dimensiuni mai mari se pot înlănțui mai multe astfel de module, criptându-se pe rând fiecare bloc de 128 biți din acesta, iar dacă dimensiunea sa nu este un multiplu a dimensiunii blocului, mesajului îi vor fi adăugați niște biți pentru a ajunge la o dimensiune adecvată.

2.3 Funcția PBKDF2

Utilizarea parolelor alese de utilizatori drept chei pentru operații criptografice nu este deloc o idee bună, datorită faptului că, de obicei, utilizatorii folosesc parole ce nu sunt destul de puternice, iar un sistem criptografic ce se bazează pe astfel de parole ar putea fi extrem de vulnerabil la atacuri de tip dictionary sau chiar prin forță brută [9].

PBKDF2 sau Password-Based Key Derivation Function 2 reprezintă o funcție folosită pentru derivarea de chei criptografice pornind de la parole alese de către utilizatori. Aceasta reprezintă o alternativă la utilizarea funcțiilor hash pentru stocarea securizată a parolelor utilizatorilor, dar este de asemenea folosită în generarea de chei secrete pentru algoritmi de criptare cu cheie simetrică de lungime fixă. Această funcție este special concepută să aibă un timp de calcul ridicat pentru a putea împiedica diverse tipuri de atacuri ce se bazează pe apeluri repetate ale acestei funcții [9].

Parametrii pe care această funcție îi primește sunt următorii [9]:

- Parola utilizatorului din care urmează să fie derivată cheia secretă
- Numărul de runde de aplicare a algoritmului asupra parolei utilizatorului
- Un salt, adică o secvență de caractere generată aleatoriu pentru a diminua riscul ca două parole să fie identice
- Lungimea cheii ce trebuie generată

Se poate deduce din prezența ultimului parametru al funcției că aceasta este capabilă să furnizeze un rezultat cu dimensiune variabilă, spre deosebire de funcțiile hash ce au întotdeauna aceeași dimensiune a valorii calculate, indiferent de dimensiunile datelor de intrare. Totodată, se pot genera chei secrete pentru algoritmi criptografici cu cheie simetrică de orice dimensiune, cum ar fi versiunile de AES-128, AES-192 sau AES-256.

Utilitatea și securitatea PBKDF2 sunt evidențiate prin faptul că această funcție criptografică este folosită în numeroase sisteme informatice, precum protejarea parolelor utilizatorilor în sistemul de operare iOS, în sistemul de criptare bazat pe AES oferit de WinZip, dar și în rețelele de internet wireless de tip WiFi protejate [9].

Capitolul 3. Arhitectura aplicației

Acest capitol cuprinde descrierea structurală și tehnică a aplicației dezvoltate. Vor fi prezentate abordările alese pentru design-ul acesteia, atât din punctul de vedere al entităților, a modului în care acestea interacționează și sunt manipulate de către aplicație, dar și a bazelor de date folosite, cât și rolurile pe care utilizatorii finali ai aplicației le pot avea.

3.1 Arhitectura MVC

Pentru a avea o organizare a codului cât mai bună și pentru o posibilitate de menținere și ușurință de dezvoltare, am ales să abordez principiul separării după structura Model-View-Controller. În plus, framework-ul Spring ales pentru dezvoltarea back-end-ului aplicației se pretează perfect pe această abordare oferind elemente ajutătoare pentru implementarea modelelor și a controller-elor.

3.1.1 Model

Aplicația dezvoltată de mine cuprinde mai multe modele necesare funcționalităților pe care acestea le pune la dispoziție. Modelele cuprind entitățile prezente în aplicație, dar și repository-uri și servicii necesare manipulării acestor entități.

În ceea ce privește implementarea entităților, framework-ul Spring oferă o modalitate abstractizată și simplificată pentru definirea entităților, oferind o serie de adnotări, precum @Table și @Entity pentru datele ce vor fi stocate în baze de date relaționale, cum ar fi MySQL, dar și @Document pentru datele stocate în MongoDB. Aplicația utilizează atât o bază de date MySQL pentru informațiile legate de utilizatori și cele necesare sistemului de prietenie implementat, cât și o bază de date MongoDB pentru datele specifice conversațiilor și a mesajelor din acestea.

Entitățile definite în cadrul aplicației sunt următoarele: User, CharRoom, ChatMessage, FriendRequest, Stats și VideoRoom. Pentru fiecare entitate definită există câte un repository necesar pentru interacțiunea cu baza de date. De asemenea, există și o serie de servicii ce implementează logica aplicației, validări ale datelor și operații mai complexe pe care un repository nu le-ar putea face de unul singur.

Toate entitățile conțin un câmp denumit id, ce este de tip Long pentru entitățile stocate în baza de date MySQL, iar pentru cele stocate în MongoDB, tipul de date al atributului id este

String. Acest câmp este folosit atât pentru identificarea unică a fiecărei instanțe a unei entități, dar și pentru a stabili relații între tabelele aferente entităților reținute în MySQL.

Entitatea User conține informații specifice contului fiecărui utilizator. Datele specifice acestei entități sunt salvate în tabelul users din baza de date MySQL și sunt gestionate prin interfața UserRepository. Entitatea are definite mai multe câmpuri specifice:

- String **username** – reprezintă numele utilizatorului;
- String **firstName** și String **lastName** – prenumele și numele de familie al utilizatorului;
- String **email** – adresa de email introdusă de către utilizator la momentul înregistrării;
- String **passwordHash** – parola utilizatorului căreia i-a fost adăugat în prealabil un salt generat în mod aleatoriu și care a fost hash-uită înainte de stocarea sa în baza de date, pentru a avea o securitate ridicată a conturilor utilizatorilor;
- boolean **isAdmin** – reține dacă utilizatorul respectiv are privilegii de administrator în aplicație sau nu;
- String **directoryName** – numele directorului asociat utilizatorului curent; fiecare utilizator are asociat un director care este creat în momentul înregistrării pentru a stoca diverse informații ce nu pot fi inserate direct în baza de date, precum poza de profil a acestuia;
- String **confirmationToken** – codul care a fost trimis pe mail-ul utilizatorului după înregistrare și care este folosit pentru verificarea contului înainte ca acesta să poată fi folosit;
- Integer **emailConfirmationTrials** – numărul de mail-uri ce au fost trimise la adresa utilizatorului pentru activarea contului;
- String **publicKey** – cheia publică asociată utilizatorului, necesară la criptarea și decriptarea folosind algoritmul RSA;
- String **encryptedPrivateKey** – cheia privată a utilizatorului, necesară pentru algoritmul RSA, ce este criptată și alături de care se reține și alte informații necesare decriptării acesteia în aplicația ce rulează pe dispozitivul utilizatorului.

Această entitate mai conține și alte câmpuri, necesare pentru stabilirea relațiilor cu celelalte entități, ce vor fi ilustrate în secțiunea de descriere a bazei de date.

Entitatea FriendRequest reprezintă cererile de prietenie pe care utilizatorii le pot trimite

unul celuilalt pentru a stabili relații de prietenie și de a putea comunica pe conversații private. Datele asociate acestei entități sunt stocate în tabelul friend_requests din baza de date MySQL și sunt gestionate cu ajutorul interfeței FriendRequestRepository. Atributele acestei entități sunt următoarele:

- User **sender** – utilizatorul ce a trimis cererea de prietenie;
- User **receiver** – utilizatorul căruia îi este destinată cererea de prietenie;
- LocalDateTime **sentAt** – momentul exact în care cererea de prietenie a fost trimisă.

Stats reprezintă o entitate ce este folosită pentru gestionarea statisticilor despre aplicație pe care administratorii le pot accesa. Acestea sunt stocate, de asemenea în baza de date MySQL și sunt gestionate de StatsRepository. Câmpurile componente ale acestei entități sunt:

- int **month** și int **year** – luna și anul pentru care sunt reținute statisticile respective;
- int **newUsersCount**, int **videoRoomsCount**, int **groupChatsCount**, int **privateChatsCount** – contorizează numărul de utilizatori ce și-au activat contul, de camere video, de conversații de grup, respectiv de conversații private create în cursul lunii și anului asociate.

Entitatea ChatRoom este destinată manipulării informațiilor despre conversațiile din aplicație, atât pentru conversațiile de tip grup, cât și pentru cele private. Datele aferente acestei entități sunt stocate în baza de date MongoDB, iar comunicarea cu baza de date se realizează prin interfața ChatRoomRepository. Atributele ce fac parte din această entitate sunt:

- String **roomName** – numele conversației, folositoare în special în cazul conversațiilor de grup;
- int **maxUsers** – numărul maxim de utilizatori ce pot lua parte la conversația respectivă; prin intermediul acestui câmp se face diferența între conversațiile private, ce pot avea maxim 2 membri, în timp ce conversațiile de grup pot conține până la 50 de membri;
- String **directoryPath** – calea către directorul asociat conversației respective, unde se pot stoca fișiere ce nu pot fi inserate în baza de date;
- Map<Long, String> **encryptedKeys** – realizează asocierea între id-urile utilizatorilor și cheile specifice fiecărei conversații, ce sunt criptate și alături de care se salvează și alte informații necesare decriptării mesajelor trimise în

conversație; pe baza valorilor din acest map se stabilește cine este membru în conversația respectivă;

- List<Long> **admins** – listă cu id-urile utilizatorilor ce au rol de administrator al conversației respective.

Această entitate conține, pe lângă câmpurile listate anterior, și câteva metode folositoare pentru manipularea datelor acesteia:

- String **getEncryptedKeyOfUser**(Long userId) – folosită pentru accesarea cheii secrete a conversației, care a fost criptată pentru utilizatorul al cărui id este furnizat în lista de parametri;
- void **setEncryptedKeyOfUser**(Long userId, String encryptedKey) – folosită pentru adăugarea cheii conversației, criptată pentru utilizatorul al cărui id este dat drept argument; practic acesta este modul în care un utilizator este adăugat în conversație;
- void **removeUserEncryptionKey**(Long userId) – utilizată pentru ștergerea cheii asociate utilizatorului primit ca parametru; aceasta este metoda de eliminare a unui utilizator dintr-o conversație;
- Long **getOtherUserId**(Long userId) – utilizată pentru obținerea id-ului celuilalt membru al unei conversații private decât cel al cărui id este dat ca parametru;
- boolean **isUserAdmin**(Long userId) – funcție ce verifică dacă utilizatorul al cărui id este dat ca parametru deține privilegii de administrator al conversației respective;
- boolean **isPrivateChat**() – funcție ce verifică dacă conversația respectivă este una privată sau este o conversație de grup.

Pentru implementarea și gestionarea conversațiilor private și de grup m-am inspirat dintr-un videoclip de pe platforma YouTube [3], de unde am preluat unele idei pe care le-am adaptat conform cerințelor aplicației mele.

ChatMessage este numele entității din aplicație ce reprezintă mesajele trimise în conversațiile utilizatorilor, fie că acestea sunt private sau de grup. Asemenea cu ChatRoom, și datele aferente acestei entități sunt stocate în baza de date MongoDB, acestea fiind manipulate prin intermediu ChatMesageRepository. Această entitate cuprinde:

- String **chatRoomId** – id-ul conversației din care face parte mesajul respectiv;

- String **contentOrPath** – conținutul criptat al mesajului, în cazul în care acesta este de tip text sau calea către imaginea trimisă, în cazul în care mesajul este de tip imagine;
- ChatMessageType **type** – tipul mesajului respectiv (TEXT, IMAGE, FILE);
- Long **senderId** – id-ul utilizatorului ce a trimis mesajul respectiv;
- Boolean **isEdited** – reține dacă mesajul respectiv a fost editat de către cel ce l-a trimis sau nu;
- Boolean **isDeleted** – marchează mesajul respectiv drept șters sau nește;
- LocalDateTime **sentAt** – momentul exact la care a fost trimis mesajul;
- List<Long> **seenBy** – lista cu id-urile membrilor conversației ce au văzut mesajul respectiv.

Ultima entitate, și anume VideoRoom, este folosită pentru gestionarea camerelor în care utilizatorii pot viziona videoclipuri pe YouTube sincronizat. Această entitate nu este persistată în nici o bază de date, întrucât camerele pot exista doar cât timp server-ul este pornit, deci stocarea de date despre aceste camere nu ar avea sens. Câmpurile ce fac parte din entitatea VideoRoom sunt:

- String **connectionCode** – codul de conectare la camera respectivă, ce este folosit pentru a identifica în mod unic entitatea, întrucât nu pot coexista două camere cu același cod de conectare;
- Set<User> **connectedUsers** – mulțime ce conține utilizatorii conectați la camera respectivă.

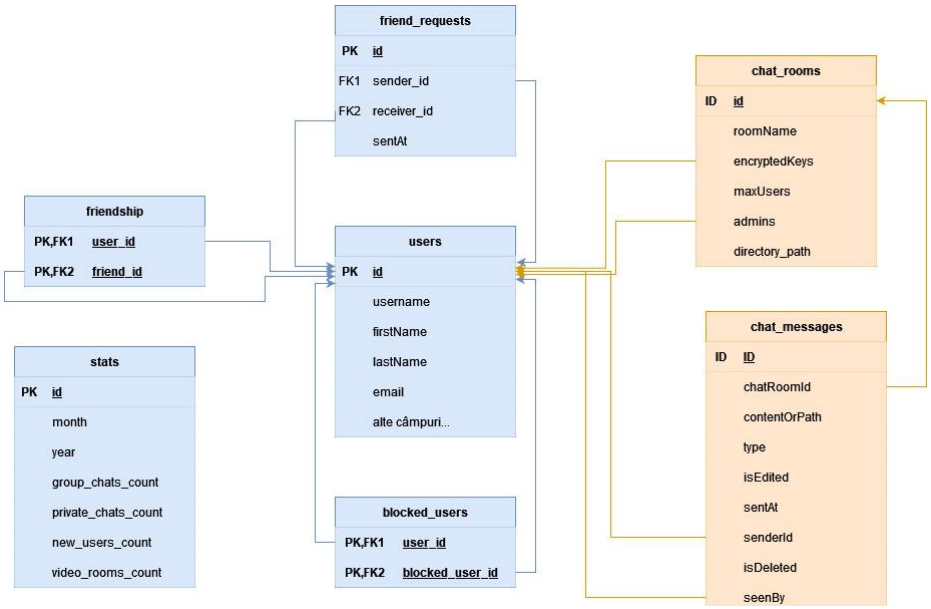

Figura 3.1 Diagrama bazei de date

Figura 3.1, de pe pagina anterioară, cuprinde diagrama bazelor de date folosite, pentru a ilustra cât mai clar legăturile dintre acestea. Chenarele albastre reprezintă tabelele din baza de date MySQL, iar cele portocalii simbolizează documentele din MongoDB. Se observă că entitatea centrală a aplicației este User, la care sunt conectate aproape toate celelalte entități, sub o formă sau alta. Singura entitate izolată este Stats, întrucât aceasta are un singur rol, acela fiind contorizarea elementelor ce iau parte la statisticile accesibile de către administratorii aplicației.

Ambele baze de date sunt rulate în containere Docker pentru evitarea instalării programelor necesare acestora pe calculatorul propriu, dar și pentru a avea un mediu izolat și ușor de replicat pentru ele. Persistența datelor este asigurată între rulările containerelor de câte un volum Docker asociat fiecărei imagini Docker. În plus, orchestrare containerelor Docker se realizează folosind unealta docker-compose.

Aplicația gestionează entitățile mai sus menționate prin clase de tip serviciu, destinate să efectueze anumite operații specifice, mai complexe decât cele implementate în repository-uri. Lista de mai jos cuprinde toate serviciile definite în partea de back-end a aplicației, dar și o scurtă descriere a acțiunilor pe care fiecare le efectuează.

- **UserService**: include metode pentru căutarea unui singur utilizator după mai multe criterii, pentru obținerea tuturor utilizatorilor cu sau fără un anumit criteriu de căutare, acordarea sau revocarea drepturilor de administrator al aplicației dar și pentru gestionarea cheilor specifice utilizatorului în baza de date;
- **AuthService**: cuprinde o serie de funcții pentru înregistrare utilizatorilor, autentificarea lor, dar și pentru inițierea trimiterii sau retrimiterii mail-urilor pentru confirmarea contului și efectuarea operației de confirmare propriu-zisă;
- **MailService**: definește o metodă generică de trimitere a unui email folosind un șablon HTML predefinit în cadrul aplicației populat cu date primite drept argumente, pe care mai apoi o folosește în definirea unei metode ce trimite un email de activare a contului utilizatorului;
- **FriendshipService**: îndeplinește sarcini precum gestionarea cererilor de prietenie, a relațiilor de prietenie și utilizatorilor blocați;
- **FileService**: în această clasă sunt incluse metode pentru crearea directoarelor aferente fiecărui utilizator și fiecărei conversații, pentru încărcarea pozelor de profil ale utilizatorilor, dar și a imaginilor criptate din conversațiile private sau de grup;

- **JWTService**: serviciul este responsabil pentru emiterea și decodarea token-ilor necesari autentificării utilizatorilor;
- **ChatRoomService**: operațiile efectuate de acest serviciu sunt cele de creare de conversații private și de grup, de adăugare sau ștergere au membrilor unei conversații, gestionarea conversațiilor unui anumit utilizator, inclusiv părăsirea unei conversații, dar și acordarea și revocarea de privilegii de administrator al unei conversații sau accesarea utilizatorilor ce sunt sau nu într-o anumită conversație;
- **ChatMessageService**: serviciul definește metode pentru trimiterea, editarea, ștergerea și recuperarea de mesaje într-o anumită conversație, dar și pentru a găsi mesajele trimise într-o conversație anume sau marcarea unor mesaje ca fiind văzute de către un anumit membru al conversației;
- **VideoRoomService**: acest serviciu este cel ce se ocupă cu gestionarea tuturor aspectelor legate de camerele de vizionare a videoclipurilor de pe YouTube în mod sincronizat; el cuprinde metode precum crearea unei astfel de camere, conectarea sau părăsirea unei camere și pentru verificarea periodică a camerelor în care nu se mai află utilizatori, pentru a le putea șterge după 5 minute;
- **StatsService**: operațiile efectuate de acest serviciu cuprind incrementarea numerelor ce constituie statisticile pe care administratorii aplicației le pot accesa, găsirea statisticilor pentru o anumită lună dintr-un an specificat, dar și a statisticilor după un anumit moment specificat sau pentru ultimele 6 luni.

3.1.2 Controller

Framework-ul Spring facilitează definirea de controlere, punând la dispoziția dezvoltatorilor numeroase adnotări precum @Controller sau @RestController. Acestea marchează clasa asociată ca fiind un bean și realizează configurări necesare pentru ca acesta să accepte cereri de la aplicațiile client. Aplicația mea conține mai multe astfel de clase, fiecare din ele expunând mai multe endpoint-uri pentru a permite aplicațiilor client să apeleze API-ul dezvoltat în back-end.

Pentru verificarea identității utilizatorilor am ales să folosesc autentificarea cu JWT (JSON Web Token) în care am reținut unele detalii despre utilizatorul autentificat. Am definit un filtru, asociat anumitor controlere, ce interceptează cererile HTTP venite de la clienți și

verifică prezența header-ului “Authorization”, de unde preia acest JWT și extrage din el informațiile necesare pe care le salvează în zona de memorie pe care firul de executare ce procesează cererea o poate accesa. De asemenea, când un client cere stabilirea unei conexiuni de tip Websocket, acesta trebuie să se autentifice cu același JWT, iar canalul de comunicare este creat cu succes doar dacă autentificarea reușește. În cazul stabilirii conexiunii Websocket, detaliile despre utilizator sunt salvate în atributele sesiunii de conectare.

Lista cu fiecare controler definit în aplicația mea, alături de o descriere pentru fiecare din acestea este următoarea:

1. **UserController**: este conceput să intercepteze cererile ce conțin prefixul ”/user” și conține dependențe către UserService și FileService. Endpoint-urile definite în acest controler sunt:
   - **GET /search –** acceptă parametrul searchString și întoarce o listă cu utilizatorii al căror nume complet sau nume de utilizator conțin valoarea primită;
   - **GET /searchNotRelated –** asemănător cu endpoint-ul anterior, însă lista întoarsă de acesta nu conține utilizatorii care au orice fel de legătură cu apelantul, adică nu există nici o cerere de prietenie, relație de prietenie sau status de blocat între apelant și nici un utilizator din această listă;
   - **POST /uploadProfilePicture –** folosit pentru încărcarea unei noi fotografii de profil pentru utilizatorul autentificat; imaginea este primită sub forma unui parametru de tipul MultipartFile;
   - **GET /profilePicture –** întoarce fotografia de profil pentru contul al cărui nume de utilizator este dat de parametrul username;
   - **GET /profilePictureById –** la fel ca endpoint-ul anterior, dar primește id-ul utilizatorului;
   - **POST /uploadKeys –** încarcă în baza de date cheile de criptare și decriptare asociate utilizatorului apelant ce sunt utilizate pentru algoritmul RSA;
   - **GET /getKeys –** întoarce perechea ce conține cheia publică și cea privată asociată apelantului;
   - **GET /getPublicKeyOfUser/{username} –** întoarce cheia publică aferentă numelui utilizatorului primit drept valoare în URL-ul cererii.

2. **AuthController**: gestionează cereri ce încep cu ”/auth”, specifice operațiilor de înregistrare a utilizatorilor, autentificare și trimitere a mail-urilor de activare a

contului. Singura referință pe care această clasă o conține este una către AuthService, acesta efectuând toate operațiile necesare în endpoint-urile acestui controller, care sunt următoarele:

- **POST /register –** preia din corpul cererii informațiile necesare și apelează serviciul pentru inserarea unei noi înregistrări în tabelul de utilizatori, dar apelează și funcția de trimitere a unui mail pentru activarea contului;
- **POST /login –** primește datele utilizatorului în corpul cererii și autentifică utilizatorul cu ajutorul serviciului;
- P**OST /confirmEmail –** verifică adresa de email a utilizatorului, folosindu-se de codul de activare primit în corpul cererii;
- **GET /getRemainingEmailConfirmationTrials** – întoarce numărul de email-uri de verificare pe care utilizatorul le mai poate cere în cazul în care nu a primit sau a pierdut mail-urile anterioare, preluând adresa de email din parametrul de interogare email;
- **POST /resendConfirmationEmail –** retrimite email-ul de activare al contului către adresa trimisă în corpul cererii;

3. **FriendshipController**: conține metode pentru gestionarea cererilor de prietenie, a relațiilor de prietenie, dar și a statusurilor de utilizatori blocați, acceptând cereri cu prefixul ”/friendship”. În ceea ce privește dependențele acestui controler, acesta necesită două servicii definite în aplicație și anume FriendshipService și ChatRoomService, dar și un obiect de tipul SimpMessagingTemplate pentru trimiterea de mesaje prin Websocket către utilizatori. Acest controller expune o serie de endpoint-uri:
   - **GET /receivedRequests –** întoarce lista cu cererile de prietenie primite de către utilizatorul apelant;
   - **GET /sentRequests** – întoarce o listă cu cererile de prietenie trimise de către utilizatorul apelant;
   - **GET /friends –** întoarce lista cu prietenii utilizatorului ce apelează acest endpoint;
   - **GET /blockedUsers –** caută și întoarce o lista cu utilizatorii pe care apelantul i-a blocat;
   - **GET /areUsersBlocked/{userId1}/{userId2} –** verifică dacă utilizatorul cu id-

ul userId1 l-a blocat pe utilizatorul cu id-ul userId2 sau vice-versa și întoarce o valoare booleană corespunzătoare;

- **POST /sendRequest/{receiver} –** trimite o cerere de prietenie din partea utilizatorului apelant către cel cu numele de utilizator dat de variabila receiver, dacă nu există o altă relație (cerere de prietenie neacceptată, relație de prietenie sau status de blocat) între cei doi;
- **POST /acceptRequest/{sender} –** acceptă cererea de prietenie trimisă de către sender utilizatorului apelant, dacă aceasta există;
- **POST /blockUser/{userToBlock} –** blochează utilizatorul userToBlock pentru apelant;
- **DELETE /rejectRequest/{sender} –** respinge cererea de prietenie pe care utilizatorul sender i-a trimis-o apelantului;
- **DELETE /cancelRequest/{receiver} –** șterge cererea pe care apelantul i-a trimis-o utilizatorului receiver;
- **DELETE /removeFriend/{friendToRemove} –** elimină relația de prietenie dintre utilizatorul friendToRemove și cel ce apelează endpoint-ul;
- **DELETE /unblockUser/{userToUnblock} –** endpoint pentru deblocarea utilizatorului userToUnblock pentru utilizatorul ce îl apelează;

4. **ChatRoomController**: definește endpoint-uri necesare pentru manipularea conversațiilor de tip chat, private și de grup, acceptând cereri ce prezintă prefixul ”/chatRoom”. Controlerul conține referințe către UserService pentru accesare informațiilor legate de utilizatori, ChatRoomService pentru gestionarea conversațiilor și ChatMessageService pentru gestionarea unor aspecte legate de mesajele din conversații prin următoarele endpoint-uri:
   - **GET /myChats –** întoarce o listă cu detaliile conversațiilor din care utilizatorul apelant face parte, dar și ultimul mesaj fiecare conversație;
   - **GET /chatUsers/{chatRoomId} –** endpoint-ul ce caută și întoarce o listă cu detalii despre utilizatorii ce fac parte dintr-o anumită conversație, al cărei id este preluat din URL-ul cererii;
   - **GET /getChatDetailsById/{chatRoomId} –** întoarce un obiect ce conține detaliile despre conversația cu id-ul chatRoomId;
   - **GET /getChatRoomKey/{chatRoomId} –** întoarce informațiile de criptare a

mesajelor în conversația respectivă (mai multe detalii în secțiunea 3.3.2)

- **GET /friendsWithNoPrivateChat –** întoarce o listă cu prietenii utilizatorului ce apelează endpoint-ul și care nu au o conversație privată cu acesta;
- **GET /friendsNotInChat/{chatRoomId} –** întoarce o listă cu prietenii utilizatorului apelant care nu se află în conversația de grup specificată prin chatRoomId;
- **POST /createPrivate/{anotherUsername} –** creează o conversație privată între utilizatorul ce apelează endpoint-ul și cel cu numele de utilizator anotherUsername;
- **POST /createGroup –** creează o conversație de grup pentru care numele și id-urile membrilor sunt date în corpul cererii HTTP;
- **POST /addUser –** adaugă un utilizator la o conversație, id-urile celor două entități fiind specificate în corpul cererii; doar administratorii conversației specificate pot apela acest endpoint; de asemenea se trimit și informațiile necesare utilizatorului introdus pentru a putea trimite și citi mesajele criptate în cadrul conversației (mai multe detalii în secțiunea 3.3.2);
- **POST /makeAdmin/{chatRoomId}/{userId} –** acest endpoint acordă utilizatorului cu id-ul userId drepturi de administrator al conversației cu id-ul chatRoomId; doar un administrator de conversație poate să adauge utilizatori;
- **DELETE /removeUser/{chatRoomId}/{userId} –** endpoint pe care doar administratorii conversației cu id-ul chatRoomId îl pot apela și care îl elimină pe utilizatorul cu id-ul userId din conversația respectivă;
- **DELETE /leaveChat/{chatRoomId} –** endpoint ce realizează operația de părăsire a conversației cu id-ul chatRoomId de către utilizatorul ce îl apelează;
- **DELETE /removeAdmin/{chatRoomId}/{userId} –** administratorii conversației cu id-ul chatRoomId pot apela acest endpoint pentru a revoca drepturile de administrator de conversație ale altui membru din aceasta, specificând id-ul respectivului utilizator.

5. **ChatMessageController:** este un controler diferit față de cele prezentate anterior, deoarece majoritatea metodelor pe care acesta le conține sunt concepute să primească mesaje prin Websocket și să trimită mai departe astfel de mesaje. Conține referințe către servicii prezentate anterior și definite de mine cum ar fi UserService,

ChatRoomService și ChatMessageService, dar și către un serviciu disponibil în framework-ul Spring și anume SimpMessagingTemplate, destinat trimiterii de mesaje către diferite canale de comunicare de tip Websocket. Endpoint-urile definite în acest controler sunt:

- **MessageMapping /sendMessage/{chatRoomId} –** primește mesajele trimise la destinația respectivă și le salvează sub formă de entități ChatMessage, utilizând atât id-ul conversației, numele de utilizator al celui ce a trimis mesajul, dar și corpul mesajului în care se află conținutul criptat, dar și tipul mesajului trimis de utilizator; după această procesare, trimite mesaje la destinațiile ”/queue/chatRoom/{chatRoomId}” și ”/queue/chatRoomUpdates” pentru a semnala trimiterea cu succes a mesajului și pentru a actualiza interfața utilizatorului instantaneu;
- **MessageMapping /seeMessage/{messageId} –** utilizat pentru marcarea unui singur mesaj, specificat prin messageId, drept văzut de către cel ce trimite mesajul Websocket; toți abonații la destinațiile ”/queue/chatRoom/{chatRoomId}” și ”/queue/chatRoomUpdates” sunt notificați de această operație pentru a marca mesajul respectiv ca văzut;
- **MessageMapping /seeAllMessages/{chatRoomId} –** se primește id-ul conversației și sunt marcate drept văzute toate mesajele din aceasta pe care utilizatorul ce trimite mesajul Websocket nu le-a văzut încă; din nou sunt notificați tot abonații la destinațiile ”/queue/chatRoom/{chatRoomId}” și ”/queue/chatRoomUpdates; acest endpoint este conceput pentru a fi apelat în momentul în care un utilizator accesează o conversație în aplicația mobilă, astfel considerându-se că a văzut toate mesajele din aceasta;
- **MessageMapping /editMessage/{messageId} –** se editează mesajul cu id-ul messageId, atribuindu-i un conținut nou, doar dacă mesajul respectiv este de tip TEXT și a fost trimis de utilizatorul salvat în sesiunea conexiunii Websocket; se trimite mesajul modificat către utilizatorii abonați la destinația ”/queue/chatRoom/{chatRoomId}” pentru a actualiza mesajul în lista de mesaje a fiecăruia;
- **MessageMapping /deleteMessage/{messageId} –** se marchează ca șters mesajul cu id-ul messageId, doar dacă acesta a fost trimis de către utilizatorul salvat în sesiunea conexiunii Websocket; se trimite mai departe un obiect ce

conține datele actualizate ale mesajului către cei ce sunt abonați la destinația ”/queue/chatRoom/{chatRoomId}” pentru a actualiza și afișa faptul că mesajul respectiv a fost șters;

- **MessageMapping /restoreMessage/{messageId} –** este setat câmpul mesajului cu id-ul messageId la fals pentru a indica faptul că mesajul nu mai este șters de către emițătorul său; este trimisă actualizarea la destinația adecvată pentru actualizarea interfeței utilizatorilor;
- **GET /chatMessages –** acesta este un simplu endpoint HTTP în care se returnează o listă cu mesajele conversației al cărei id este specificat prin parametrul chatRoomId ce au fost trimise înainte de data primită prin parametrul beforeTimestampStr; este folosit pentru cererea mesajelor într-o manieră secvențială, trimițându-se maximum 50 de mesaje per cerere.

6. **VideoRoomController:** gestionează operațiile de creare și alăturare unei camere de vizionare de videoclipuri, conținând referințe către servicii definite de către mine, care sunt UserService și VideoRoomService, dar și către un obiect de tipul SimpMessagingTemplate, necesar trimiterii de mesaje prin Websocket. Acestea sunt cele două endpoint-uri definite în cadrul controlerului:
   - **POST /createNew** – creează o cameră nouă de vizionare de videoclipuri în cadrul serviciului VideoRoomService și întoarce detaliile camerei proaspăt create;
   - **POST /join/{connectionCode} –** adaugă utilizatorul ce apelează endpoint-ul în camera de vizionare de videoclipuri aferentă codului de conectare furnizat în URL-ul cererii, iar dacă operația reușește trimite un semnal de acutalizare a listei de membri ai camerei către toți utilizatorii ce sunt abonați la destinația ”/queue/videoRoom/joinOrLeave/{connectionCode}”.

7. **VideoRoomMessageController**: toate endpoint-urile expuse de către acest controler sunt de tip MessageMapping și gestionează acțiunile ce se întâmplă în timp real în cameră. În ceea ce privește dependențele de care acest controler are nevoie, acestea constau în serviciile UserService și VideoRoomService și în șablonul SimpMessagingTemplate. Endpoint-urile acestui controller sunt:
   - **MessageMapping /videoRoom/syncVideo/{connectionCode} –** primește un

mesaj fără corp și trimite către utilizatorii abonați la destinația ”/queue/videoRoom/signal/{connectionCode}” un semnal de tip SYNC_VIDEO ce reprezintă cererea de sincronizarea a videoclipului din momentul în care un utilizator se alătură camerei;

- **MessageMapping /videoRoom/syncVideoResponse/{connectionCode} –** mesajul primit de acest endpoint conține id-ul videoclipului de pe YouTube ce este rulat la un moment dat în camera cu codul ce conectare connectionCode; în acest endpoint se trimite un nou mesaj Websocket către utilizatorii abonați la destinația ”/queue/videoRoom/signal/{connectionCode}” de tip SYNC_VIDEO_RESPONSE cu id-ul videoclipului primit;
- **MessageMapping /videoRoom/syncPosition/{connectionCode} –** prin acest endpoint se face o cerere de sincronizare a poziției din videoclipul care rulează în camera de vizionare cu codul de conectare connectionCode la un moment dat; se trimite mai departe un semnal de tipul SYNC_POSITION către destinația ”/queue/videoRoom/signal/{connectionCode}”;
- **MessageMapping /videoRoom/syncPositionResponse/{connectionCode} –** endpoint destinat trimiterii poziției curente a player-ului video din camera cu codul de conectare connectionCode și primește în corpul mesajului Websocket un obiect ce conține poziția respectivă sub formă de String și o variabilă booleană ce indică dacă videoclipul este oprit sau nu; se trimite mai departe un semnal către destinația ”/queue/videoRoom/signal/{connectionCode}” de tipul SYNC_POSITION_RESPONSE care include detaliile primite în mesajul Websocket;
- **MessageMapping /videoRoom/pause/{connectionCode} –** endpoint ce preia codul ce conectare și trimite un semnal de tipul PAUSE către destinația ”/queue/videoRoom/signal/{connectionCode}” pentru a declanșa oprirea pentru fiecarui utilizator conectat la camera respectivă;
- **MessageMapping /videoRoom/resume/{connectionCode} –** se preia codul de conectare și se trimite un semnal de tip RESUME către toți utilizatori abonați la destinația ”/queue/videoRoom/signal/{connectionCode}” pentru a porni videoclipul după o oprire anterioară;
- **MessageMapping /videoRoom/seekToPosition/{connectionCode} –** primește un mesaj al cărui corp conține poziția la care se mută videoclipul și o

valoarea booleană ce indică starea player-ului (oprit sau pornit); trimite mai departe un semnal de tipul SEEK_TO_POSITION către destinația aferentă camerei respective pentru vizionare de videoclipuri ce conține informațiile primite;

- **MessageMapping /videoRoom/changeVideo/{connectionCode} –** gestionează schimbarea videoclipurilor, primind un obiect în care se specifică id-ul noului videoclip și trimițând mai departe un semnal de tipul CHANGE_VIDEO către cei ce sunt abonați la destinația asociată camerei cu codul de conectare dat pentru a încărca noul videoclip în player-ul fiecărui utilizator;
- **MessageMapping /videoRoom/leave/{connectionCode} –** se preiau datele utilizatorului din sesiunea Websocket, care va fi eliminat din camera pentru vizionare de videoclipuri și se va trimite un semnal pentru actualizarea listei de membri către participanții ce încă sunt conectați la această cameră.

8. **AdminController**: acest controler include endpoint-uri HTTP accesibile doar de către utilizatorii ce au permisiuni de administrator al aplicației. Cererile procesate de către acest controler încep cu ”/admin”. Dependențele necesare acestui controler sunt UserService și StatsService pentru gestionarea utilizatorilor dar și vizualizarea statisticilor în aplicație. Lista endpoint-urilor este următoarea:
   - **GET /getAllUsers** – întoarce o listă cu obiecte ce conțin detalii despre utilizatori și rolul acestora în aplicație;
   - **GET /getStats –** întoarce o listă de obiecte de tip Stats ce conțin statisticile legate de numărul de conturi activate, numărul de conversații private și de grup create și numărul de camere pentru vizionare de videoclipuri deschise în cursul ultimelor 6 luni calendaristice;
   - **POST /makeAdmin/{userId} –** oferă privilegii de administrator al aplicației utilizatorului cu id-ul userId;
   - **POST /removeAdmin/{userId} –** revocă drepturile de administrator pentru utilizatorul cu id-ul specificat în URL-ul cererii HTTP.

3.1.3 View

Cea de-a treia componentă a abordării MVC adoptate este cea de View. Aceasta are poate cel mai important rol pentru utilizatorul final, întrucât stabilește modul în care acesta interacționează cu aplicația. View-ul trimite cereri către endpoint-urile din controlerele definite la nivelul server-ului, care, la rândul lor, apelează servicii pentru a manipula datele din model, iar în final răspunsul este afișat utilizatorului final tot prin intermediul view-ului.

În cazul proiectului meu, view-ul constă într-o aplicație în platforma Android, construită cu ajutorul framework-ului Flutter. Acesta prezintă o serie de widget-uri, oferind posibilitatea construirii unei interfețe interactive și intuitive. Aplicația mea utilizează numeroase astfel de widget-uri, atât din pachete ce fac parte din biblioteca implicită ce vine alături de Flutter, cum ar fi MaterialApp, Scaffold sau ListView, dar și din unele pachete ce necesită instalare ulterioară, precum CachedNetworkImage, YoutubePlayer sau GoRouter.

În spatele interfeței pe care utilizatorul o poate vedea pe ecranul propriului dispozitiv mobil stă o serie de clase ce interacționează pentru a pune la dispoziție funcționalitățile specifice aplicației. Pentru un proces de dezvoltare și întreținere a codului cât mai facil, acestea sunt structurate în mai multe categorii cu denumiri descriptive.

Directorul ”components” conține clase de tip Widget ce reprezintă componente ce sunt reutilizate în mai multe părți ale aplicației și personalizate în funcție de caz prin argumentele constructorilor lor.

Pentru a beneficia de verificare a tipurilor statică și sugestii în cadrul IDE-ului folosit pentru dezvoltare, în directorul ”interfaces” sunt definite clase asociate obiectelor primite din apelurile către back-end. Astfel, după parsarea corpului răspunsului primit de la server rezultă un obiect de tip Map<String, dynamic> ce poate fi transformat într-un obiect al unei clase de tip interfață prin utilizarea constructorilor de tip factory. De aici, acest obiect poate fi manipulat ca orice obiect din limbajul de programare Dart.

Fiecare pagină (ecran) a aplicației este definită într-un fișier separat, iar toate aceste pagini sunt conținute în directorul ”pages”. Manipularea acestor pagini în ceea ce privește navigarea se face folosind pachetul GoRouter, a cărei configurații este descrisă în fișierul ”routes” din directorul ”routing”. Aici sunt definite o serie de rute, ce reprezintă niște obiecte în care se specifică calea de navigare către acestea și o funcție de tip callback folosită pentru schimbarea efectivă a paginii. Opțional se poate specifica o funcție callback pentru redirecționarea utilizatorului către alte rute în anumite cazuri specifice, cum ar fi lipsa privilegiilor de administrator al aplicației când este accesată ruta ”/admin”, moment în care

utilizatorul este redirecționat către pagina principală a aplicației.

Pentru a evita accesarea directă a endpoint-urilor din back-end în widget-uri, am ales să definesc în cadrul aplicației clase de tip serviciu ce se ocupă cu apelarea endpoint-urilor de tip HTTP, dar și cu trimiterea de mesaje pentru conexiunile de tip Websocket. Metodele din aceste clase întorc, de regulă, interfețele specifice datelor primite de la server sau un mesaj de tip String în caz de eroare pentru afișarea acestuia utilizatorului. Pentru a eficientiza aplicația din punct de vedere al memoriei, am ales folosirea unui Provider pentru majoritatea serviciilor, rolul acestuia fiind de a furniza instanțe ale claselor de tip serviciu oriunde este necesar în aplicație, fără nevoia de a crea unele noi de fiecare dată. Excepție face însă StompService, ce este implementat într-o clasă de tip Singleton și gestionează conexiunea unică de tip Websocket stabilită între aplicația de front-end și server.

Aplicația mea include o serie de clase și funcții utilitare folosite în numeroase scenarii, definite în fișierele din directorul ”utils”. Aici se pot găsi clase precum HttpWithToken ce conține metode unde este adăugat automat header-ul Authorization, ce conține token-ul JWT necesar autentificării și autorizării utilizatorilor, celor mai uzuale metode HTTP. Clasele CryptoUtils și RegexValidator, de asemenea definite în acest director, conțin metode specifice operațiilor criptografice din cadrul aplicației, respectiv validării unor secvențe folosind expresii regulate. În directorul respectiv se pot găsi și alte fișiere utilitare ce conțin detaliile de conectare la server-ul aplicației, dar și funcții pentru parsarea obiectelor de tip Duration din String.

În ceea ce privește designul interfeței utilizatorului, am construit una specifică acestui tip de aplicații, personalizând-o după cum am considerat că este mai intuitiv și mai ușor de folosit. Tema aleasă este una întunecată și este setată în fișierul principal al proiectului și anume main.dart, pentru care am definit o paletă de culori proprie, bazată pe nuanțe de albastru. Mai multe detalii, inclusiv capturi de ecran cu aspectul aplicației mobile sunt disponibile în capitolul 4.

3.2 Rolurile utilizatorilor

Existența mai multor roluri pentru utilizatori reprezintă un mod perfect de a oferi acestora diferite privilegii, fără a fi nevoie să li se atribuie anumite permisiuni în mod direct fiecăruia. Sunt definite astfel clase ce au acces la anumite funcționalități, iar utilizatorii sunt adăugați în aceste clase, moștenind astfel drepturile specifice clasei din care fac parte.

În aplicația dezvoltată de mine există mai multe tipuri de utilizatori. În funcție de

privilegiile la nivel de aplicație, utilizatorii se împart în utilizatori neautentificați, utilizatori autentificați simpli, ce pot accesa majoritatea funcționalităților aplicației și administratori ai aplicației ce au unele drepturi speciale. Pe de altă parte, din punctul de vedere al conversațiilor de tip chat, utilizatorii se împart în membri și administratori de conversație. În cele ce urmează voi prezenta fiecare dintre aceste roluri, vorbind despre funcționalitățile pe care fiecare le poate accesa.

3.2.1 Utilizatorul neautentificat

Având în vedere că aplicația mea se bazează pe aspecte legate de identitatea fiecărui utilizator, această categorie nu are acces la foarte multe funcționalități. Astfel, cei ce nu sunt autentificați au acces doar la funcționalități ce nu necesită date legate de un anumit cont și sunt asociate operațiilor de autentificare. Aceste funcționalități includ:

- Înregistrarea unui nou cont furnizând datele necesare, cum ar fi un nume de utilizator unic, numele de familie, prenumele, adresa de email și o parolă puternică pentru a spori securitatea contului;
- Activarea contului prin trimiterea codului trimis pe email în momentul creării contului;
- Cererea de retrimitere a email-ului ce conține codul de activare al contului, în cazul pierderii mail-ului inițial, în limita a 3 mail-uri trimise în total;
- Autentificarea prin trimiterea numelui de utilizator sau a adresei de email și a parolei asociate contului pentru care se realizează autentificarea.

3.2.2 Utilizatorul autentificat

După efectuarea autentificării și obținerea token-ului JWT, utilizatorul capătă statutul de autentificat. Din acest moment, acesta poate utiliza token-ul respectiv pentru a accesa diverse funcționalități, informațiile sale fiind reținute în acesta. Funcționalitățile la care un utilizator autentificat, fără privilegii de administrator al aplicației le are sunt următoarele:

- Interacțiunea cu sistemul de prieteni; aceasta include gestionarea cererilor de prietenie ce îl privesc pe utilizatorul respectiv, gestionarea relațiilor de prietenie pe care acesta le are cu alți utilizatori, dar și blocarea interacțiunii altor utilizatori sau deblocarea acestora, după caz;

- Crearea de conversații de tip chat, atât private, cât și de grup; pentru a crea o conversație, este nevoie ca toți utilizatorii pe care cineva dorește să îi adauge în aceasta să fie prieteni cu acesta;
- Schimbarea fotografiei de profil; utilizatorii autentificați pot alege o fotografie în format JPG sau PNG și o pot încărca drept fotografie de profil;
- Accesarea conversațiilor de tip chat din care utilizatorul face parte, vizualizarea mesajelor și a membrilor acestora, dar și trimiterea de noi mesaje și gestionarea mesajelor proprii;
- Părăsirea conversațiilor de grup la care utilizatorul nu mai dorește să participe;
- Crearea de camere pentru vizionarea videoclipurilor de pe YouTube sincronizat cu alți utilizatori autentificați;
- Conectarea la o cameră pentru vizionare de videoclipuri, dar și părăsirea acesteia;
- Interacțiunea cu player-ul video din camera în care utilizatorul se află; această funcționalitate include trimiterea de semnale pentru oprirea sau pornirea player-ului, mutarea la o nouă poziție din videoclip, schimbarea videoclipului și semnale de sincronizare în momentul conectării la cameră către toți ceilalți membri ai camerei pentru a păstra videoclipul sincronizat pe toate dispozitivele conectate.

3.2.3 Administrator al unei conversații

Acest rol este relevant doar în cadrul conversațiilor de tip chat, în special în cele de tip grup. Când o conversație este creată de către un utilizator, acesta devine automat primul administrator al acelei conversații. Administratorii unei conversații au acces la următoarele funcționalități în cadrul conversației respective:

- Adăugarea sau ștergerea de utilizatori din conversație; un administrator de conversație poate să adauge într-un grup doar alți utilizatori ce sunt prieteni cu acesta și care nu fac parte deja din grupul respectiv;
- Acordarea sau revocarea privilegiilor de administrator al conversației altor utilizatori; nu este permis ca un administrator de conversație să își revoce propriile privilegii.

3.2.4 Administrator al aplicației

Rolul de administrator al aplicației include toate funcționalitățile utilizatorului autentificat și se poate suprapune cu rolul de administrator de conversație, însă nu există nici o corelație între cele două. În ceea ce privește funcționalitățile pe care administratorii aplicației le pot accesa, acestea includ:

- Accesarea listei cu toate conturile active înregistrare în aplicație;
- Acordarea sau revocarea privilegiilor de administrator al aplicației altui utilizator;
- Accesarea unei liste ce conține statisticile lunare despre numărul de conturi activate, de conversații private și de grup și de camere pentru vizionare de videoclipuri create în ultimele 6 luni.

3.3 Gestionarea cheilor criptografice & criptarea mesajelor

Din moment ce o funcționalitate principală din aplicația mea constă în trimiterea de mesaje criptate ce pot fi citite doar de către membrii conversației în care au fost trimise (end-to-end encryption), a fost necesară integrarea algoritmilor criptografici prezentați în capitolul 2. Pe scurt, la crearea fiecărei conversații este creată și o cheie simetrică pentru criptarea și decriptarea mesajelor acesteia. Am decis că pentru a păstra aceste chei în baza de date ele trebuie să fie criptate la rândul lor pentru fiecare membru al conversației, folosind un sistem de criptare asimetrică. Având în vedere aceste considerente, am fost nevoit să găsesc o strategie pentru gestionarea cât mai sigură a cheilor de criptare și decriptare folosite de către algoritmii criptografici. În această secțiune vor descrie această strategie, dar și modul în care mesajele trimise în conversații sunt criptate.

3.3.1 Gestionarea cheilor pentru algoritmul RSA

O variantă a algoritmului RSA este folosită în aplicația mea pentru a cripta cheile simetrice specifice conversațiilor, personalizat pentru fiecare utilizator. Acesta este, totodată, și modul în care utilizatorii sunt adăugați în conversație, pe care îl voi descrie pe larg în secțiunea 3.3.2.

Perechea de chei este generată de către aplicația mobilă în momentul în care utilizatorul se înregistrează. Figura 3.2, de pe pagina următoare, ilustrează acțiunile petrecute în momentul înregistrării utilizatorului. Se poate observa că după ce utilizatorul a introdus datele și formatul

acestora este validat cu succes de către aplicația mobilă este generată o pereche de chei pentru algoritmul RSA. Mai apoi, din parola utilizatorului este derivată cu ajutorul funcției PBKDF2 o cheie pentru criptare simetrică, folosind un salt, dar este generat și un vector de inițializare (IV). Folosind cheia și vectorul se criptează cheia privată a utilizatorului cu algoritmul AES, iar odată cu detaliile introduse de utilizator pentru înregistrare sunt trimise și cheia publică, dar și un String numit encryptedPrivateKey ce conține reprezentările în format Base64 ale salt-ului pentru derivarea cheii din parolă, vectorului de inițializare și a cheii private criptate, toate fiind separate prin delimitatorul ”.”. Aceste informații sunt trimise la server, iar dacă nu mai există nici un cont cu numele de utilizator și email-ul introduse, se creează acest cont și informațiile sunt salvate în el.

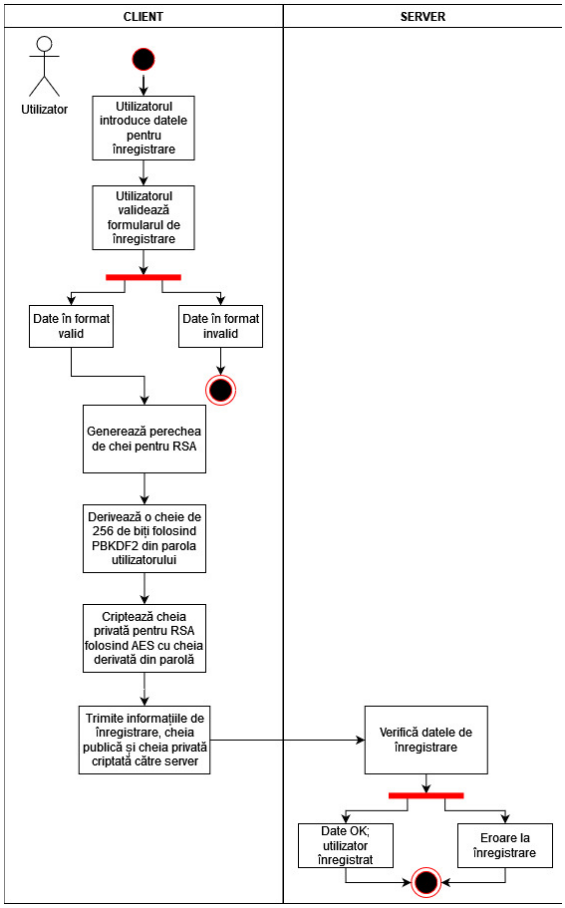

Figura 3.2 Diagrama pentru înregistrarea utilizatorului

După înregistrare și activarea contului, de fiecare dată când utilizatorul se va autentifica în aplicație vor avea loc evenimentele schițate în figura 3.3. Se vor trimite numele de utilizator sau adresa de email și parola către server, iar dacă acesta autentifică utilizatorul fără vreo excepție, este generat un JWT care este trimis alături de cheia publică și cheia privată criptată asociate contului către aplicația client. La primirea răspunsului, clientul preia salt-ul, vectorul de inițializare și cheia privată criptată și le decodează din formatul Base64. Urmează să deriveze aceeași cheie ca în momentul înregistrării cu funcția PBKDF2, din parola utilizatorului și salt-ul preluat de la server. Folosind această cheie simetrică este decriptată cheia privată prin AES, iar

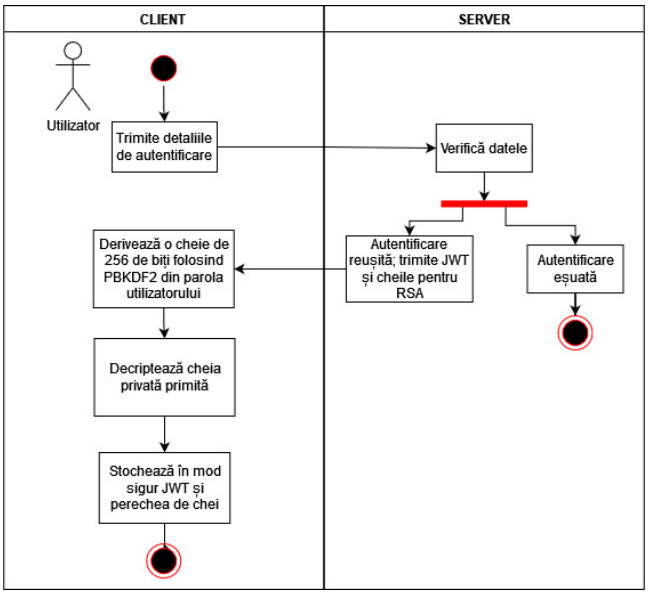

Figura 3.3 Diagrama pentru autentificarea utilizatorului

atât aceasta, cât și cheia publică și token-ul JWT sunt stocate într-o zonă sigură de memorie pe dispozitivul mobil al utilizatorului.

3.3.2 Gestionarea cheilor simetrice pentru algoritmul AES

După cum am spus în introducerea secțiunii 3.3, mesajele trimise în conversații sunt stocate criptat, iar pentru aceasta este nevoie de o cheie de criptare simetrică specifică fiecărei conversații. În paragrafele următoare voi descrie modul de gestionare al acestor chei criptografice.

Figura 3.4 ilustrează operațiile efectuate la crearea unei conversații. Când un utilizator trimite cererea de creare a unei conversații, fie ea privată sau de grup, server-ul verifică datele primite, iar dacă sunt în regulă creează obiectul de tip ChatRoom și generează o cheie simetrică de 256 biți și un vector de inițializare pentru algoritmul AES. Urmează să transforme aceste două obiecte în String-uri prin reprezentarea lor în Base64 și să le concateneze, separându-le prin ”.”. Acest String rezultat este mai apoi criptat cu fiecare cheie publică RSA a utilizatorilor ce trebuie adăugați în conversație și sunt inserate perechile formate din id-ul utilizatorului și String-ul criptat în HashMap-ul aferent conversației respective.

Procesul de adăugare a unui utilizator într-o conversație de grup constă, de asemenea în mai mulți pași, evidențiați în figura 3.5. Când un administrator al

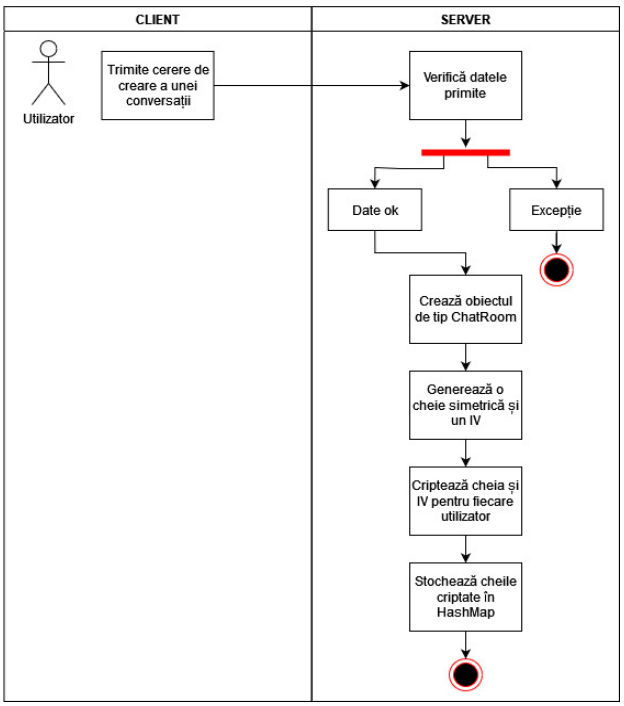

Figura 3.4 Crearea unei conversații

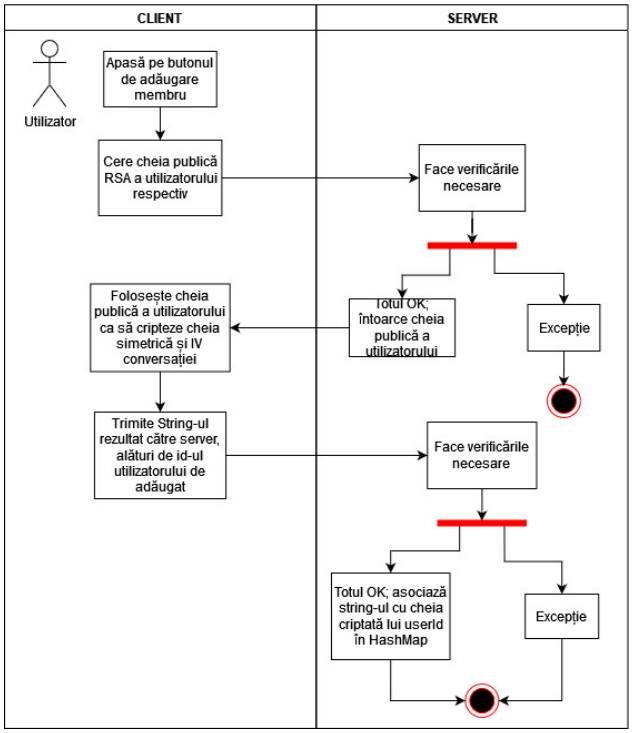

Figura 3.5 Adăugarea unui utilizator în conversație de grup

conversației respective apasă pe butonul pentru adăugarea unui nou utilizator în grup, se face un apel către server în care se cere cheia publică a utilizatorului ce se dorește a fi adăugat în conversație. Dacă totul decurge bine, aplicația client va primi în răspunsul HTTP cheia publică cerută. Aplicația de mobil preia cheia simetrică și vectorul de inițializare pentru AES stocate pe dispozitivul utilizatorului și creează String-ul ce conține reprezentarea acestor două elemente în format Bast64, separate prin ”.”. Acesta se criptează folosind cheia publică a utilizatorului primită în cererea anterioară și trimite rezultatul, alături de id-ul utilizatorului și id-ul camerei către server. Aici au loc unele verificări, urmând ca perechea formată din id-ul utilizatorului și String-ul ce conține cheia criptată a conversației să fie introduse în HashMap-ul asociat camerei respective, în cazul în care toate verificările sunt îndeplinite.

Acest HashMap ce face parte din entitatea ChatRoom are o dublă utilitate: determinarea apartenenței unui utilizator la o anumită conversație și stocarea cheii simetrice a conversației criptată special pentru utilizatorul respectiv folosind cheia sa publică. Pentru ca utilizatorii să poată trimite sau vizualiza mesaje într-o conversație, aceștia trebuie să facă parte din aceasta și să se afle în posesia cheii secrete și a vectorului de inițializare specifice acelei conversații. În acest sens, procesul a cărei diagramă este reprezentată în figura 3.6 este întocmai cel de accesare a detaliilor criptografice necesare pentru o conversație. Când utilizatorul intră într-o conversație din interfața aplicației mobile se face o verificare, iar dacă detaliile specifice conversației se găsesc stocate pe dispozitivul utilizatorului acestea sunt folosite pentru interacțiunea cu acea conversație. În caz contrar, se face un apel către server în care se cer aceste detalii. Server-ul face verificările de rigoare, iar dacă totul este ok întoarce utilizatorului valoarea asociată id-ului său din HashMap-ul obiectului de tip ChatRoom. Aplicația mobilă decriptează conținutul răspunsului folosind cheia privată pentru RSA a utilizatorului autentificat, obține din String-ul decriptat reprezentările în Base64 ale vectorului de inițializare și a cheii simetrice, și le stochează într-o zonă sigură de memorie pentru utilizare ulterioară.

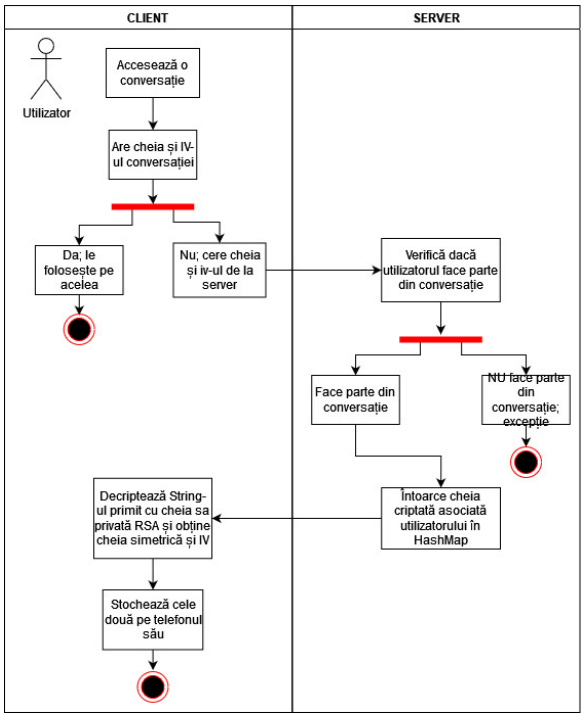

Figura 3.6 Diagrama de accesare a cheii unei conversații

3.3.3 Criptarea mesajelor din conversații

După cum am menționat și în secțiunea anterioară, mesajele din toate conversațiile, fie ele private sau de grup sunt criptate în momentul trimiterii pe dispozitivul utilizatorului cu algoritmul AES, utilizând cheia simetrică și vectorul de inițializare specifice conversației respective. Mesajul criptat este mai apoi trimis prin Websocket către server la destinația ”/app/sendMessage/{chatRoomId}”. Serverul interceptează acest mesaj și salvează acest mesaj în baza de date prin crearea unei instanțe a entității ChatMessage, după care trimite mai departe mesajul către toți utilizatorii abonați la destinația specifică conversației respective, și anume ”/queue/chatRoom/{chatRoomId}”, dar și către destinația ”/queue/chatRoomUpdates” pentru a semnala apariția unui nou mesaj. Aplicațiile mobile ale membrilor conversației respective actualizează lista de mesaje din conversație, decriptând conținutul mesajului primit cu algoritmul AES pentru a-l putea afișa, dar și lista de conversații în mod corespunzător.

În cazul editării unui mesaj procesul este unul similar cu cel descris în paragraful anterior. Noul conținut al mesajului este criptat în aplicația mobilă și trimis către server, unde mesajul salvat este actualizat și marcat drept editat, iar noua variantă a acestui mesaj este transmisă către utilizatorii care decriptează și afișează mesajul editat. Ștergerea și restaurarea mesajelor prezintă un proces similar, însă singura modificare ce are loc este marcarea mesajului ca având conținutul șters, respectiv nește pentru a nu pierde informațiile pe care utilizatorii le transmit, în cazul ștergerii mesajelor.

Anunțurile ce apar în cadrul conversației sunt tratate la rândul lor drept mesaje, singura diferență este că au o valoarea diferită a câmpului type din entitatea ChatMessage. În timp ce mesajele ce conțin ideile utilizatorilor sunt marcate cu ChatMessageType.TEXT, mesajele de tip anunț rețin în atributul type valoarea ChatMessageType.ANNOUNCEMENT. Cu toate acestea, la trimiterea și primirea mesajelor de tip anunț, conținutul lor este criptat, respectiv decriptat, în aceeași manieră ca mesajele de tip text.

3.3.4 Eliminarea cheilor criptografice

Cheile criptografice și vectorii de inițializare sunt date sensibile pe care un utilizator neautorizat le poate exploata în scopuri malițioase. Din acest motiv a trebuit să găsesc strategia de gestionare a acestora în cadrul aplicației mele. Cu toate acestea, strategia descrisă până în acest moment este incompletă, întrucât în momentul în care un utilizator iese din cont, pe dispozitivul acestuia rămân cheile RSA asociate contului precedent, dar și cheile și vectorii de

inițializare specifice fiecărei conversații din care contul respectiv făcea parte.

Din acest motiv am ales ca în momentul ieșirii din cont în cadrul aplicației mobile să se șteargă, pe lângă token-ul JWT necesar autentificării utilizatorului la server, cheile asociate contului său, dar și toate datele criptografice specifice conversațiilor. Astfel, când în aplicația mobilă nu este autentificat nici un utilizator, datele sensibile ale contului precedent nu vor fi expuse nici unui fel de pericol, iar când un nou utilizator se autentifică nu vor exista conflicte în ceea ce privește zona de memorie sigură în care sunt stocate aceste date.

Capitolul 4. Funcționalitățile aplicației

Acest capitol conține o descriere amănunțită a funcționalităților oferite de către aplicația dezvoltată, incluzând capturi de ecran pentru o mai bună înțelegere a modului de interacțiune cu aceasta.

4.1 Funcționalități legate de autentificare

În secțiunea aceasta sunt prezentate funcționalitățile pe care utilizatorii finali le pot accesa pentru autentificarea lor în cadrul aplicației. Acestea includ înregistrarea unui nou cont, activarea contului prin email și autentificarea folosind datele unui cont existent.

4.1.1 Autentificarea utilizatorilor

În momentul în care un utilizator nu este autentificat în aplicația mobilă, acesta va fi întâmpinat de ecranul pentru autentificare, ilustrat în figura 4.1. Acestuia îi este adresat un mesaj prietenos și este rugat să introducă datele de autentificare în câmpurile afișate. Pentru a putea verifica corectitudinea parolei, poate apăsa pe iconița din câmpul ”Password”, astfel încât să activeze sau dezactiveze ascunderea parolei. După introducerea datelor, utilizatorul apasă pe butonul ”Log In”, iar datele sunt trimise către server pentru validare. Procesul de autentificare este descris în detaliu în secțiuea 3.3.1. În cazul în care autentificarea eșuează, un mesaj de eroare este afișat utilizatorului, iar dacă autentificarea se realizează cu succes, detaliile primite de la server sunt salvate pe dispozitivul utilizatorului și acesta este redirecționat către pagina principală a aplicației.

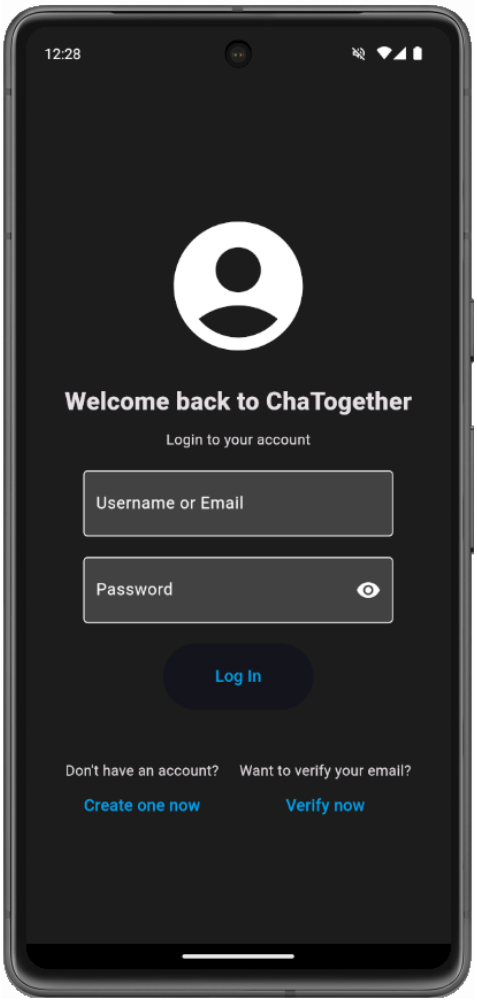

Figura 4.1 Ecranul de autentificare a utilizatorului

Din această pagină, utilizatorul are alte două variante, și anume navigarea către pagina de înregistrare

a unui nou cont sau navigarea către pagina de verificare a adresei de email pentru activarea unui cont deja înregistrat.

4.1.2 Înregistrarea unui nou cont

Ecranul din care utilizatorul poate înregistra un cont nou în aplicație este vizibil în figura 4.2. Utilizatorul este îndemnat să își creeze un cont prin completarea câmpurilor prezente în pagină. Câmpurile destinate numelui de utilizator și parolei conțin o iconiță de informare pe care utilizatorul poate apăsa lung pentru a putea vedea cerințele specifice fiecăruia. În plus, poate vizualiza sau ascunde atât valoarea câmpului pentru parolă, cât și a celui pentru confirmarea parolei prin apăsarea iconiței sub formă de ochi. După introducerea datelor și apăsarea butonului de înregistrare, formularul este validat din punct de vedere al formatului datelor, iar dacă există date în format invalid este specificat acest lucru prin evidențierea cu roșu a câmpurilor respective. Pe de altă parte, dacă validarea reușește, datele sunt trimise la server, unde are loc procesul descris în secțiunea 3.3.1. În cazul în care la server nu apare nici o excepție datorată datelor introduse de utilizator, acesta este redirecționat către pagina de autentificare, iar dacă o astfel de excepție apare, utilizatorul vede un mesaj de eroare specific.

Figura 4.2 Ecranul de înregistrare a unui cont

În josul ecranului de înregistrare există un buton, care nu apare în figura alăturată, ce redirecționează utilizatorul către pagina de autentificare, în cazul în care acesta are deja un cont și dorește să se autentifice cu acela.

4.1.3 Activarea unui cont înregistrat

Pentru a se putea autentifica și folosi majoritatea funcționalităților aplicației, un utilizator trebuie să își activeze mai întâi contul creat. Acest lucru se poate face din ecranul

dedicat acestei operații, la care poate ajunge din cel de autentificare. Figura 4.3 prezintă înfățișarea ecranului de verificare a adresei de email. Aici există un câmp în care utilizatorul trebuie să introducă un cod de 10 caractere alfanumerice primit pe email la momentul înregistrării contului. După introducerea codului și apăsarea butonului de verificare este verificat formatul acestuia, iar dacă validarea reușește, codul este trimis către server pentru activarea contului. La server are loc verificarea codului trimis cu cel salvat în baza de date, iar dacă cele două se potrivesc activarea contului este efectuată, iar utilizatorul este redirecționat către pagina de autentificare.

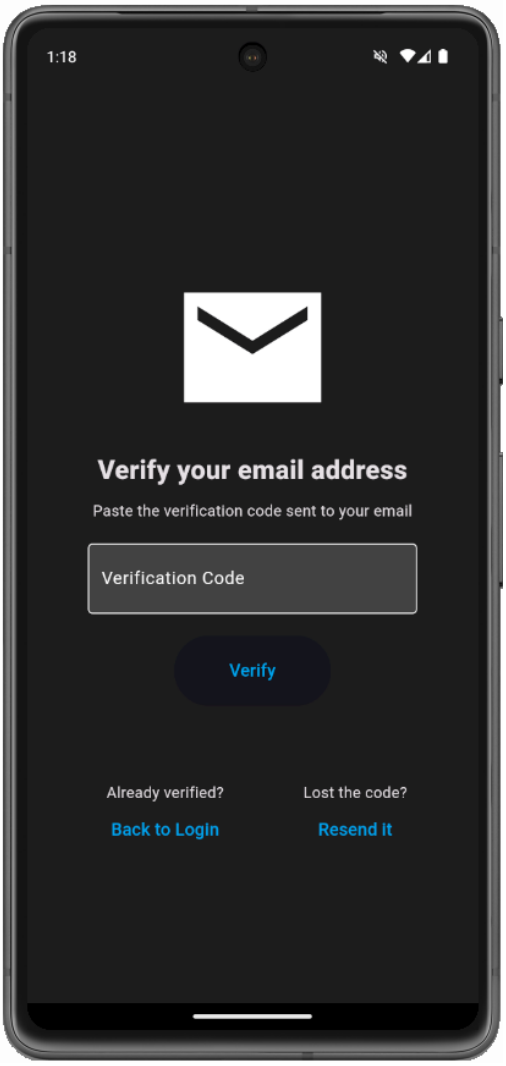

Figura 4.3 Ecranul de activare a unui cont

Tot în această pagină utilizatorul poate cere trimiterea unui nou ma il de activare a contului prin apăsarea butonului ”Resend It” și furnizarea adresei de email asociate contului respectiv. Dacă toate verificările sunt trecute, un nou mail de activare a contului este trimis la adresa specificată. În plus, în pagina de verificare a adresei de email există și un buton de întoarcere la pagina de autentificare, pentru cazul în care se dorește autentificarea cu un cont deja activat.

4.2 Funcționalități după autentificare

Această secțiune prezintă funcționalitățile la care un utilizator autentificat are acces în cadrul aplicației mobile. După o autentificare cu succes, utilizatorul este redirecționat către pagina principală a aplicației, alcătuită din trei zone principale, evidențiate în figura 4.4, de pe pagina următoare:

- Bara din partea superioară, evidențiată cu un chenar roșu, unde sunt afișate numele aplicației și imaginea de profil a utilizatorului autentificat, aceasta din urmă conducând la pagina cu detaliile utilizatorului, de unde se poate efectua și ieșirea din cont;

- Bara de navigare din partea de jos a ecranului, marcată cu albastru, marchează pagina pe care utilizatorul se află la un moment dat, dar oferă acestuia și posibilitatea de a schimba pagina prin apăsarea butoanelor vizibile; opțiunile unui utilizator ce nu are privilegii de administrator al aplicației sunt: pagina de conversații, pagina de relații cu alți utilizatori și pagina de gestionare a unei camere pentru vizionare de videoclipuri;
- Secțiunea de conținut, încadrată în chenarul galben, afișează informațiile specifice fiecărei pagini accesibile din bara de navigare; în cazul de față, utilizatorul autentificat se află pe pagina de conversații unde apare un mesaj pentru a-l anunța că nu face parte din nici o conversație momentan.

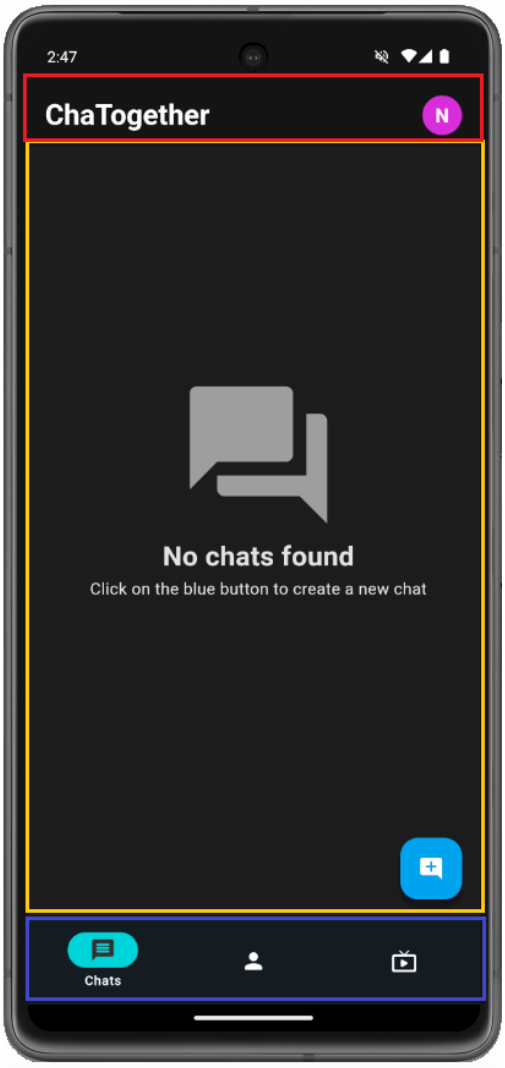

Figura 4.4 Structura ecranului principal

În plus, paginile de conversații și de relații cu alți utilizatori conțin câte un buton de tip Floating Action Button pentru inițierea unor acțiuni specifice fiecărei pagini.

4.2.1 Conversații și mesaje

Prima destinație din bara de navigare a aplicației este reprezentată de pagina de conversații, surprinsă pe pagina următoare, în figura 4.5. Aici apare o listă cu acele conversații din care utilizatorul înregistrat face parte. Fiecare element din această listă conține numele conversației (numele celuilalt utilizator în cazul conversațiilor private și numele grupului în cazul celor de grup), o imagine reprezentativă după caz și tipul de conversație (privată sau de grup). Dacă o anumită conversație conține mesaje pe care utilizatorul autentificat nu le-a văzut, atunci în dreptul acesteia apare o bulină albastră pentru a atrage atenția utilizatorului. Actualizările privind statusul ultimului mesaj, dar și a conversațiilor în care utilizatorul este adăugat sau eliminat se petrec în timp real, datorită conexiunii Websocket.

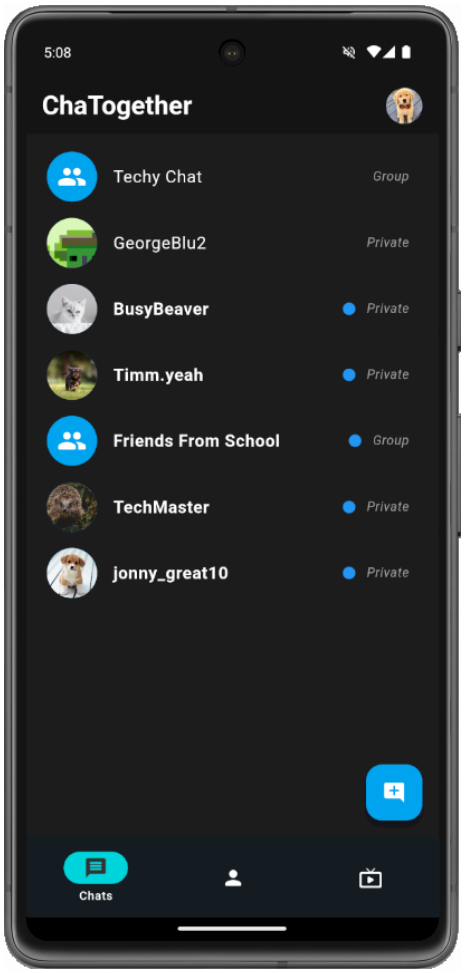

Figura 4.5 Ecranul cu toate conversațiile utilizatorului

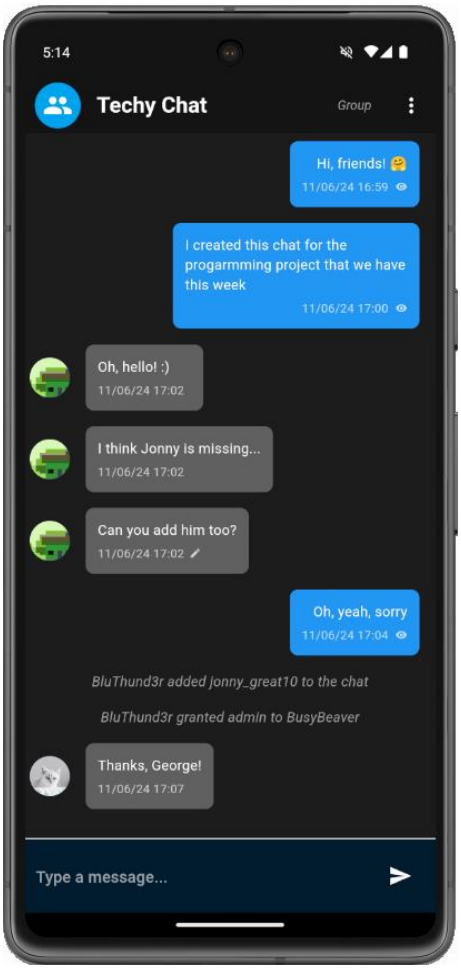

Figura 4.6 Conversație de grup

Butonul poziționat în colțul din dreapta jos al secțiunii de conținut este utilizat pentru crearea unor noi conversații. La apăsarea acestuia, utilizatorul poate alege dacă dorește să creeze o conversație privată sau de grup și este redirecționat către o pagină specială în car poate selecta utilizatorul sau utilizatorii, după caz, pentru a crea o nouă conversație. Ambele tipuri de conversații pot include, în momentul creării, doar utilizatori ce sunt prieteni direcți cu creatorul acesteia. Pentru conversația de tip grup trebuie specificat și un nume specific. În cazul ambelor tipuri de conversație, după confirmarea creării grupului, o cerere va fi trimisă către server în acest sens, iar dacă totul este în regulă, noua conversație va apărea în lista fiecărui membru.

Pentru a accesa o conversație, utilizatorul trebuie să apese pe elementul din listă asociat acesteia. Figura 4.6 este o captură de ecran dintr-o conversație de grup din lista utilizatorului. În momentul accesării unei conversații, dispozitivul se abonează la destinația ”/queue/chatRoom/{chatRoomId}” pentru a primi toate mesajele trimise din această conversație prin server. Amplasarea elementelor este una familiară pentru utilizatori, având în partea de jos un câmp pentru formularea mesajelor și un buton pentru trimiterea lor, în zona din mijloc fiind vizibile mesajele din conversație, iar în partea de sus fiind afișate numele

conversației, o iconiță corespunzătoare unui grup (în cazul conversațiilor private se afișează fotografia de profil a celuilalt utilizator, urmată de numele acestuia), tipul conversației și un buton pentru mai multe opțiuni.

Mesajele pot fi manipulate de către utilizatorul ce le-a trimis. Prin apăsarea lungă pe un astfel de mesaj sunt afișate în partea de jos a ecranului opțiuni de editare, copiere, iar după caz apare și opțiunea de ștergere sau restaurare a unui mesaj. În cazul apăsării lungi pe mesajele ce nu au fost trimise de utilizatorul autentificat, acestuia îi este afișată doar opțiunea de copiere a conținutului mesajului.

Prin apăsarea pe cele trei puncte prezente în colțul din dreapta sus se pot accesa acțiuni ce privesc conversația. Administratorii conversației au acces la a vizualiza membrii conversației, de unde pot elimina alți utilizatori sau gestiona drepturile lor în cadrul conversației, de a adăuga alți prieteni de-ai lor ce nu fac parte deja din conversație și de a părăsi conversația. Membrii conversației ce nu dețin astfel de privilegii au doar opțiunile de a vedea lista întreagă cu toți utilizatorii ce iau parte la conversație și de a o părăsi. La fiecare acțiune specifică administratorilor de conversație, dar și în momentul în care un membru o părăsește, este trimis un mesaj de tip anunț pentru a-i notifica pe ceilalți despre starea curentă. În cazul conversațiilor private, ambii participanți dețin rol de utilizator, însă singura acțiune ce privește conversația este cea de blocare sau, după caz, deblocare a celuilalt utilizator. În momentul blocării, nici un utilizator nu mai poate trimite mesaje către acea conversație privată, fiind afișat un mesaj sugestiv în locul câmpului pentru compunerea mesajelor. Când un utilizator îl deblochează pe celălalt, acest câmp revine instantaneu amândurora, iar ambii pot discuta în continuare.

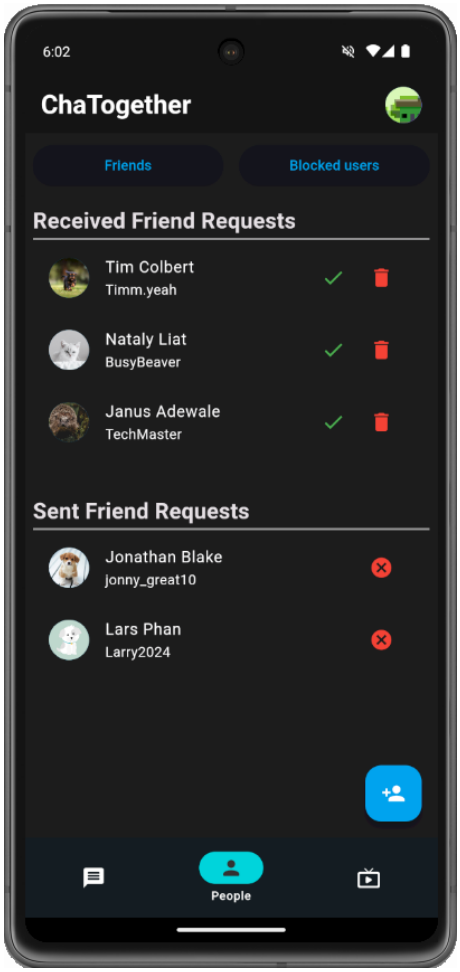

Figura 4.7 Pagina de relații cu alți utilizatori

4.2.2 Relațiile cu alți utilizatori

Utilizatorii autentificați au acces și la sistemul de prietenii din cadrul aplicației, dar și la blocarea altor utilizatori. În figura 4.7 se poate observa interfața acestei pagini, în care utilizatorul poate să vizualizeze cererile de prietenie primite, dar și trimise, cu care poate interacționa

într-un mod intuitiv prin acceptarea sau respingerea cererilor primite, dar și prin ștergerea cererilor trimise.

Butonul de tip Floating Action Button conduce către o nouă pagină, în care utilizatorul poate căuta după numele de utilizator sau numele complet o altă persoană către care să trimită o cerere de prietenie prin apăsarea butonului ”Send Request”, alăturat fiecărui utilizator afișat după căutare.

Cele două butoane: ”Friends” și ”Blocked Users” ce apar în partea de sus a ecranului conduc utilizatorul către pagina cu prieteni, respectiv către pagina cu utilizatorii blocați ai utilizatorului autentificat. Cele două pagini menționate au o structură asemănătoare, conținând o bară de căutare în partea de sus, necesară filtrării listei de utilizatori afișați ce se află chiar sub aceasta. În pagina cu prieteni utilizatorul poate să vizualizeze prietenii săi din aplicație și să îi șteargă după cum consideră, în timp ce pagina cealaltă conține lista utilizatorilor blocați, de unde, totodată, se pot debloca utilizatorii cu care se dorește a interacționa direct.

4.2.3 Camere pentru vizionare de videoclipuri

Cea de-a treia destinație din bara de navigare este destinată gestionării camerelor pentru vizionare de videoclipuri de pe YouTube sincronizate. Figura 4.8 constă într-o captură de ecran în care sunt surprinse elementele paginii respective. Această pagină este una simplă, ce conține două butoane și text pentru îndrumarea utilizatorului.

Butonul ”Create Room” trimite către server cererea de creare a unei noi camere de vizionare. După ce server-ul procesează cererea primită și întoarce un răspuns ce conține detaliile camerei create, inclusiv codul de conectare , aplicația mobilă folosește acest cod pentru a cere conectarea la camera creată. Dacă nu apare nici o situație excepțională, utilizatorul este condus către pagina specifică unei camere de vizionare, personalizată pentru camera respectivă.

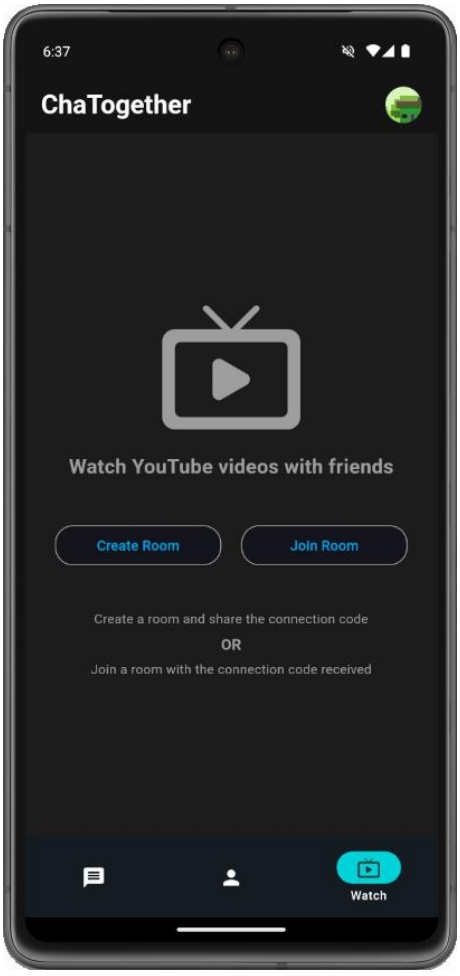

Figura 4.8 Ecranul cu opțiuni pentru camerele de vizionare videoclipuri

Dacă utilizatorul alege să se alăture unei camere de vizionare deja existente, acesta va apăsa butonul ”Join Room” și va fi nevoit să introducă cele 6 cifre aferente codului de conectare la aceasta. În cazul în care codul respectiv este corect, utilizatorul se va alătura camerei într-un proces similar celui descris în paragraful precedent.

Figura 4.9 surprinde structura paginii aferente camerei de vizionare. Se poate observa cum în partea de sus este afișat codul de conectare la această cameră pe care membrii pot apăsa pentru a-l pot copia și distribui către alte persoane. În dreapta acestui cod se află butonul de părăsire al camerei. Mai jos de acestea se află în această ordine: playerul video în care vor rula videoclipurile, un câmp în care utilizatorii pot pune URL-uri către videoclipuri de pe YouTube, alături de un buton pentru rularea videclipului respectiv, iar la final se află secțiunea în care sunt afișați membrii camerei de vizionare.

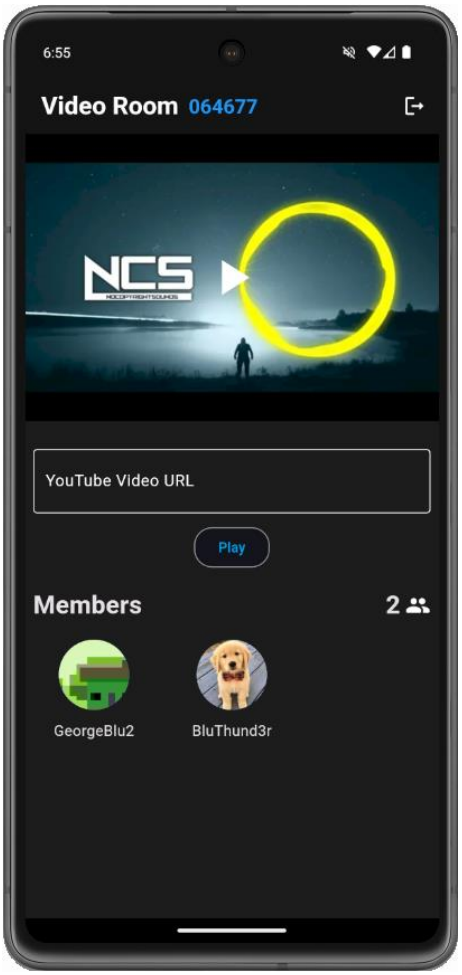

Figura 4.9 Ecranul pentru camera de vizionare

În momentul conectării la cameră, aplicația mobilă trimite prin Websocket cereri către ceilalți membri pentru a obține id-ul videoclipului care rulează și poziția din videoclip la care a ajuns playerul. Dacă în cameră este doar un membru (cel ce tocmai a intrat), nu se trimit aceste semnale. Odată sincronizați, toți membrii pot interacționa cu playerul prin oprirea, pornirea sau mutarea la o altă poziție din videoclip, acțiunile având efect și pentru playerele celorlalți membri . De asemenea prin introducerea unui URL valid către un videoclip de pe YouTube în câmpul de sub player, urmată de apăsarea butonului ”Play”, se va încărca pentru toți membri un nou videoclip în acest player. În plus, playerul poate să fie mărit pentru fiecare utilizator, în mod independent, astfel încât să ocupe întregul ecran, oferind o experiență de vizionare cât mai plăcută și fără alte distrageri.

Lista membrilor este actualizată în timp real, datorită conexiunii Websocket, pe măsură ce noi membri se alătură camerei de vizionare sau când un utilizator părăsește camera respectivă. O cameră este activă pentru aproximativ 5 minute din momentul în care lista de membri devine goală. Dacă trec 5 minute fără ca alt utilizator să se conecteze la cameră, aceasta este ștearsă, însă procesul se întrerupe și camera poate fi folosită în continuare în cazul în care

un utilizator se conectează până la scurgerea timpului.

4.3 Panoul de administrare al aplicației

Utilizatorii cu rol de administrator al aplicației au acces la o destinație în plus din bara de navigare ce conduce către panoul de administrare al aplicației, surprins în figura 4.10. Această pagină conține două butoane ce redirecționează administratorul către pagina de statistici, respectiv de utilizatori.

Figura 4.11 ilustrează pagina cu statistici în care administratorul poate vizualiza datele statistice lunare legate de numărul de conturi activate, conversații private și de grup create, dar și de camere video inițiate în ultimele 6 luni. Folosind butoanele poziționate în partea de sus a paginii, el poate schimba modul de vizualizare între grafice și o listă cu datele respective, grupate după luna în care sunt înregistrate.

Pagina din figura 4.12 este cea de utilizatori, în care un administrator poate vedea toate conturile înregistrate în aplicație, activate sau nu. De asemenea, poate căuta un anumit cont după numele de utilizator sau numele complet, iar la apăsarea pe un anumit cont, poate vizualiza detaliile acestuia și acorda sau revoca drepturi de administrator utilizatorului respectiv.

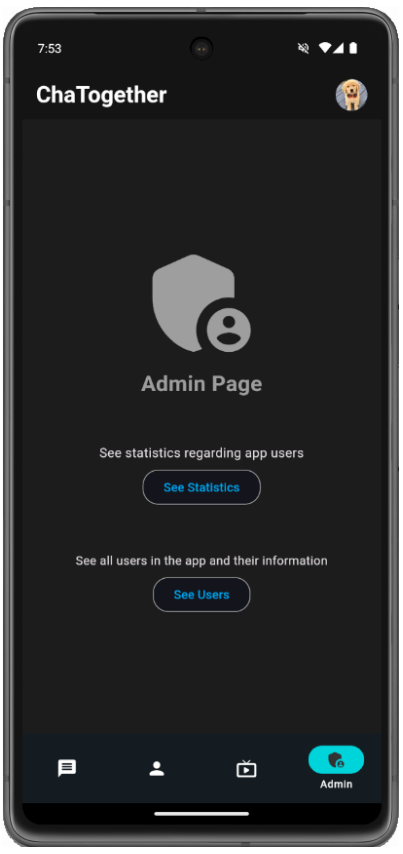

Figura 4.10 Panoul de administrare

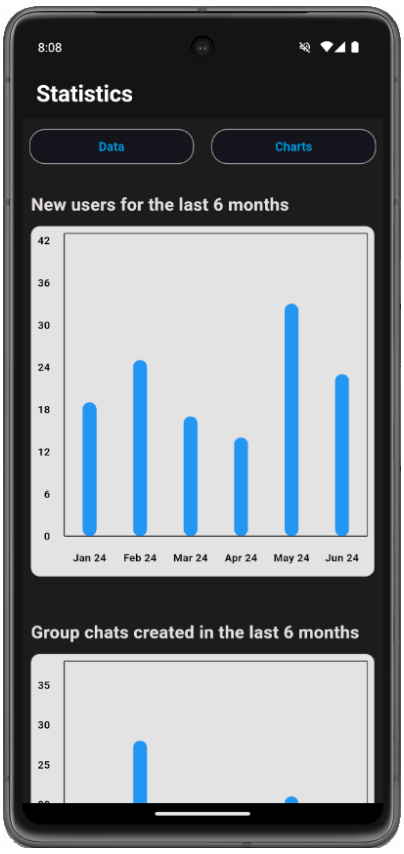

Figura 4.11 Ecranul de statistici

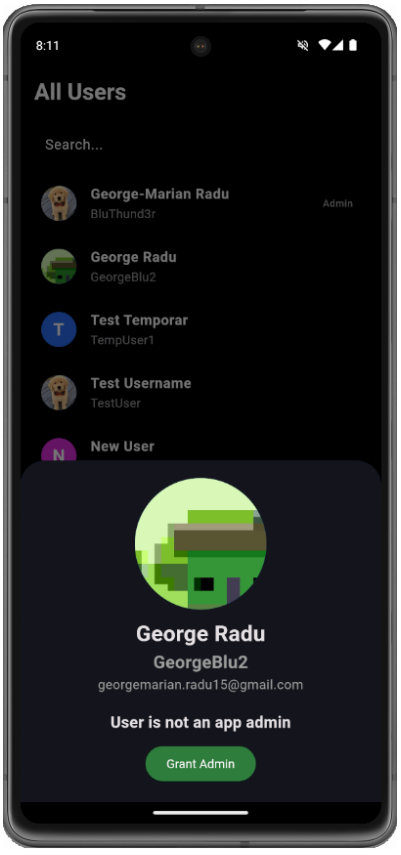

Figura 4.12 Ecranul de utilizatori

Capitolul 5. Testarea aplicației

Pentru a valida faptul că aplicația se comportă întocmai modului cum este proiectată este necesară realizarea testării acesteia. Testarea software este în sine o ramură a dezvoltării aplicațiilor software și conține mai multe abordări, dintre care cea mai populară în rândul programatorilor este testarea funcțională. Această strategie constă în specificarea unor date de intrare și a comportamentului așteptat pe care elementul testat trebuie să îl îndeplinească. În cazul în care comportamentul așteptat este identic cu cel real, testul se consideră trecut, iar în caz contrar testul eșuează.

Pentru testarea aplicației mele am ales să implementez o suită de teste unitare ce vor valida funcțional comportamentul unor componente în mod izolat, fără a acoperi modul în care acestea interacționează. Testele au fost implementate folosind biblioteca JUnit [23].

5.1 Simularea bazelor de date

Este foarte important ca testele să nu interacționeze cu bazele de date folosite în mediul de dezvoltare și de producție pentru a nu altera conținutul deja existent în aplicație. Pentru a asigura acest lucru, am ales două abordări diferite, câte una pentru fiecare bază de date utilizată în aplicația mea.

5.1.1 Simularea bazei de date MySQL

În ceea ce privește simularea bazei de date relaționale MySQL pe care o utilizez, a fost necesar să integrez pentru task-ul de testare al uneltei de build folosite o nouă bază de date, H2, reținută în memoria procesului de server. Aici am făcut configurările necesare pentru ca framework-ul Spring să se integreze cu această nouă bază de date, acesta ocupându-se și de definirea schemei bazei de date H2 prin ierarhia de clase de tip entitate. Această schemă din păcate nu conține date, iar pentru acest lucru a fost nevoie să creez un fișier numit ”data.sql” în care să scriu cereri SQL pentru inserarea de noi date folosite la testare. Starea bazei de date din memorie poate fi modificată în cadrul testelor la fel cum se modifică și baza de date principală, însă după fiecare test se asigură revenirea la starea inițială, definită în fișierul menționat anterior.

5.1.2 Simularea bazei de date MongoDB

În cazul bazei de date MongoDB nu am putut proceda la fel ca pentru cea MySQL, întrucât cele două sunt diferite în natura lor, prima fiind o bază de date cu date stocate în documente și colecții, iar cea de-a doua fiind una relațională. Aici am ales să folosesc un server MongoDB integrat (embedded) oferit de către pachetul flapdoodle. Prin intermediul acestui pachet am configurat un nou server MongoDB cu aceeași versiune ca cea din mediul de dezvoltare, însă ce rulează pe un alt port pentru a nu avea conflicte cu aplicația. Practic acest server va fi pornit înaintea executării grupării de teste ce au nevoie să interacționeze cu baza de date NoSQL și va fi oprit după ce aceste teste vor fi rulate.

5.2 Testele Unitare

Testele unitare implementate asigură comportamentul așteptat pentru o parte din componentele principale ale aplicației. Astfel, am creat 4 suite de teste, fiecare testând funcționalitatea unui serviciu din aplicația mea:

- ChatRoomService: pentru acest serviciu am definit câte un caz în care funcțiile de creare a unei conversații private și a uneia de grup să reușească în a crea tipul respectiv de conversație, dar și câteva cazuri în care se așteaptă ca acestea să nu poată crea conversațiile;
- FriendshipService: testarea acestui serviciu a constat în verificarea că funcția de trimitere a unei cereri de prietenie este executată cu succes când datele furnizate respectă logica acesteia, dar și a faptului că în anumite cazuri, precum cel în care utilizatorii sunt deja prieteni, printre altele, executarea acestei funcții este oprită de o excepție specifică; de asemenea a fost testată și funcția de acceptare a unei cereri de prietenie, fiind acoperite mai multe cazuri de eșuare a funcției, de exemplu cel în care cererea de prietenie nu există sau cei doi utilizatori sunt deja prieteni;
- VideoRoomService: aici am testat atât funcția ce creează o cameră video nouă pentru cazul de executare cu succes, întrucât nu ar exista un caz specific de eșec, cât și pe cea de alăturare unei camere, acoperind cazul de succes și cele de eșec în care codul de conectare specificat nu este asociat cu nici o cameră sau cel în care utilizatorul deja este conectat la camera dorită;
- StatsService: ultimul serviciu testat este cel ce se ocupă cu gestionarea statisticilor pe care administratorii aplicației le pot accesa; aici am creat teste ce verifică întoarcerea corectă a listei de statistici pentru ultimele 6 luni calendaristice, dar și

pentru incrementarea numărului de conturi noi, conversații private și de grup, dar și al camerelor video create în ultima lună.

Prin suita de teste definite se asigură astfel faptul că metodele ce au fost acoperite funcționează conform așteptărilor, că pot fi folosite pentru dezvoltarea altor funcționalități ce se bazează pe ele, dar și că modificările ulterioare aduse elementelor testate nu alterează comportamentul așteptat al acestora.

Concluzie

Aplicația ChaTogether, dezvoltată în platforma Android, oferă funcționalități diverse pentru împlinirea scopului final, și anume posibilitatea utilizatorilor de a socializa în mod sigur atât în conversații private, cât și de grup și de a viziona videoclipuri sincronizate de pe unul dintre cele mai populare website-uri de video streaming, YouTube. Toate aceste funcționalități sunt puse la dispoziția utilizatorilor finali printr-o interfață minimalistă, modernă și intuitivă pentru a face experiența acestora cât mai plăcută în cadrul aplicației.

Proiectul este unul amplu ce combină mai multe subdomenii ale informaticii. De la proiectarea și dezvoltarea unui RESTful API alături de construirea unei interfețe intuitive pentru utilizator, la implementarea unei comunicări în timp real folosind protocolul Websocket și chiar integrarea algoritmilor criptografici pentru a asigura un grad ridicat de securitate în cadrul conversațiilor, aplicația se dovedește a fi complexă și de încredere pentru utilizatorii săi.

Asemenea oricărui alt proiect software, și aplicația mea prezintă anumite limitări și funcționalități ce nu au fost implementate în totalitate sau chiar deloc, dar probabil că pe măsură ce nevoile utilizatorilor se schimbă, vor trebui adăugate în aplicație. Câteva astfel de direcții viitoare de dezvoltare pot fi:

- Includerea apelurilor audio și video pentru o comunicare mai facilă și mai eficientă;
- Posibilitatea trimiterii de imagini criptate end-to-end în conversațiile de tip chat;
- Opțiunea de a personaliza conversațiile de grup prin schimbare numelui sau a imaginii reprezentative;
- Implementarea unui sistem de recomandări al persoanelor posibil cunoscute de către utilizatorul autentificat;
- Trimiterea de email-uri de activare al contului din panoul de administrare către utilizatorii ce au pierdut sau nu au primit cele 3 email-uri trimise automat;
- Îmbunătățirea performanței prin stocarea unor copii ale datelor (mesaje și imagini) pe dispozitivul utilizatorului;

În final, experiența de dezvoltare a acestei aplicații m-a ajutat să îmi îmbunătățesc aptitudinile de programator, dar, totodată, am avut ocazia de a aplica pentru prima dată anumite concepte studiate la cursurile din cadrul facultății. În plus, am putut lucra cu tehnologii de actualitate și noi pentru mine, descoperind noi abordări și moduri de gândire ce îmi vor fi cu siguranță de folos în proiectele viitoare.

Bibliografie

[1] A. M. Abdullah, *Advanced Encryption Standard (AES) Algorithm to Encrypt and Decrypt Data*, Eastern Mediterranean University – Cyprus, publicat în iunie 2017, de la pagina a doua până la pagina a patra. Accesibil la adresa:

https://www.researchgate.net/profile/Ako-Abdullah/publication/317615794_Advanced_Encryption_Standard_AES_Algorithm_to_Encrypt_and_Decrypt_Data/links/59437cd8a6fdccb93ab28a48/Advanced-Encryption-Standard-AES-Algorithm-to-Encrypt-and-Decrypt-Data.pdf (ultima accesare 12.06.2024)

[2] A. Morales, *The 25 greatest Java apps ever written*, publicat la data 1.05.2020. Accesibil la adresa: https://blogs.oracle.com/javamagazine/post/the-top-25-greatest-java-apps-ever-written (ultima accesare 12.06.2024)

[3] B. Ali, *WebSocket Tutorial with Spring Boot | Build One On One Chat Application*, canalul de Youtube Bouali Ali, publicat în noiembrie 2023. Accesibil la adresa:

https://www.youtube.com/watch?v=7T-HnTE6v64&list=PLFJAQC35dNcIJdTuU_CkARjgO1ovHN-MP&index=5 (ultima accesare 12.06.2024)

[4] B. J. Evans, J. Clark, D. Flanagan, *Java in a Nutshell*, O'Reilly, publicat în februarie 2023, capitolul 1. Accesibil la adresa:

https://books.google.ro/books?hl=ro&lr=&id=K-GtEAAAQBAJ&oi=fnd&pg=PT11&dq=what+is+java&ots=KdTITNxuqa&sig=QsakQmarKaPosheOkontcrigIyo&redir_esc=y#v=onepage&q&f=false (ultima accesare 12.06.2024)

[5] E. Milanov, *The RSA Algorithm*, Universitatea din Washington, publicat în iunie 2009, paginile 1-6. Accesibil la adresa:

https://sites.math.washington.edu/~morrow/336_09/papers/Yevgeny.pdf (ultima accesare 12.06.2024)

[6] E. Windmill, *Flutter in Action*, Manning Publications, publicat în ianuarie 2020, capitolul 1. Accesibil la adresa:

https://books.google.ro/books?hl=ro&lr=&id=EzgzEAAAQBAJ&oi=fnd&pg=PT19&dq=flutter&ots=7zoGF43sD3&sig=DkiFGLFp6gyPW5t12mqlD3YYS2g&redir_esc=y#v=onepage&q=flutter&f=false (ultima accesare 12.06.2024)

[7] G. Bracha, *The Dart Programming Language*, Addison-Wesley Professional, publicat în decembrie 2015, capitolul 1. Accesibil la adresa:

https://books.google.ro/books?hl=ro&lr=&id=UHAlCwAAQBAJ&oi=fnd&pg=PT13&dq=dart+programming+language&ots=Pq_Rh0F8GU&sig=eooe5olfnTanEfyPTy-ij_6gQNM&redir_esc=y#v=onepage&q&f=false (ultima accesare 12.06.2024)

[8] K. Banker, D. Garrett, P. Bakkum, S. Verch, *MongoDB in Action: Covers MongoDB version 3.0*, Manning Publications, publicat în martie 2016, capitolul 1 până la secțiunea 1.2.5. Accesibil la adresa:

https://books.google.ro/books?hl=ro&lr=&id=kzkzEAAAQBAJ&oi=fnd&pg=PT21&dq=mongo+db&ots=8V1_uU5187&sig=anj7bdaxIDPbU5YKO58QcxfepEw&redir_esc=y#v=onepage&q=mongo%20db&f=false (ultima accesare 12.06.2024)

[9] L. Ertaul, M. Kaur, V. A. K. R. Gudise, *Implementation and Performance Analysis of PBKDF2, Bcrypt, Scrypt Algorithms*, CSU East Bay California USA, fără dată de publicare, de la prima pagină până la a treia pagină. Accesibil la adresa:

http://mcs.csueastbay.edu/~lertaul/PBKDFBCRYPTCAMREADYICWN16.pdf (ultima accesare 12.06.2024)

[10] N. H. Freese, D. C. Norris, A. E. Loraine, *Integrated genome browser: visual analytics platform for genomics*, Oxford Academic, Bioinformatics vol. 32, publicat în aprilie 2016. Accesibil la adresa: https://doi.org/10.1093/bioinformatics/btw069 (ultima accesare 12.06.2024)

[11] V. Wang, F. Salim, P. Moskovits, *The Definitive Guide to HTML5 WebSocket*, Apress Berkeley, publicat în februarie 2013, pagina 36.

[12] Y. B. Chika, O. K. Esther, *Financial stock application using websocket in real time application*, International Journal of Informatics and Communication Technology vol. 8 no. 3, publicat în decembrie 2019, pagina 144. Accesibil la adresa: https://ijict.iaescore.com/index.php/IJICT/article/view/20254/12843 (ultima accesare 12.06.2024)

[13] Despre java garbage collector, site-ul oficial Oracle:

https://www.oracle.com/webfolder/technetwork/tutorials/obe/java/gc01/index.html (ultima accesare 12.06.2024)

[14] Documentația oficială Java 8: https://www.oracle.com/java/technologies/javase/8-whats-new.html (ultima accesare 12.06.2024)

[15] Documentația oficială a framework-ului Spring, versiunea 3.2.17, accesibilă la adresa: https://docs.spring.io/spring-framework/docs/3.2.17.RELEASE/spring-framework-reference/pdf/spring-framework-reference.pdf (ultima accesare 12.06.2024)

[16] Documentația oficială Spring Boot, versiunea 3.3.0, accesibilă la adresa: https://docs.spring.io/spring-boot/ (ultima accesare 12.06.2024)

[17] Documentația oficială a limbajului Dart versiunea 3.4: https://dart.dev/overview (ultima accesare 12.06.2024)

[18] Documentația oficială Docker: https://docs.docker.com/get-started/overview/ (ultima accesare 12.06.2024)

[19] Pagina oficială a proiectului BioJava: https://biojava.org/wiki/About (ultima accesare 12.06.2024)

[20] Platforma pentru statistici legate de utilizatorii lunari de Minecraft: https://activeplayer.io/minecraft/ (ultima accesare 12.06.2024)

[21] Prezentarea MySQL de pe site-ul official Oracle: https://www.oracle.com/mysql/what-is-mysql/ (ultima accesare 12.06.2024)

[22] Site-ul oficial dedicat framework-ului Flutter: https://docs.flutter.dev/ (ultima accesare 12.06.2024)

[23] Site-ul oficial cu manualul de utilizare JUnit: https://junit.org/junit5/docs/current/user-guide/ (ultima accesare 14.06.2024)