

Programul de studii:

Informatică

Lucrare de licență

Aplicație socială pentru studenți

StudentConnect

|  |  |
| --- | --- |
| Autor: | **Matache David Mihai** |
| Coordonator ştiințific: | **Conf. Dr. Deaconu Adrian** |

Braşov, 2022

**Cuprins**

[1.Introducere 5](#_Toc106901859)

[1.2 Scopul 8](#_Toc106901860)

[1.2.1 Obiective 8](#_Toc106901861)

[2. Tehnologiile software folosite 9](#_Toc106901862)

[2.1 JetBrains IntelliJ IDEA 9](#_Toc106901863)

[2.2 Programare în Java 10](#_Toc106901864)

[2.3 Android 10](#_Toc106901865)

[2.3.1 Sistemul de operare Android 10](#_Toc106901866)

[2.3.2 Dezvoltare de software în mediul Android 11](#_Toc106901867)

[2.3.2.1 Android SDK 11](#_Toc106901868)

[2.3.2.2 „Android Studio IDE” 12](#_Toc106901869)

[2.3.2.3 Componentele unei aplicați Android 13](#_Toc106901870)

[2.3.2.3.1 Componentele fundamentale 13](#_Toc106901871)

[2.3.2.3.2 Manifest file 14](#_Toc106901872)

[2.3.2.3.3 Resursele aplicației 15](#_Toc106901873)

[2.3.2.4 Limbajul XML 15](#_Toc106901874)

[2.3.2.4.1 XML în Android Development 15](#_Toc106901875)

[2.4 „SQL” 16](#_Toc106901876)

[2.5 gRPC 18](#_Toc106901877)

[2.5.1 Protocol Buffers 18](#_Toc106901878)

3. [Aplicația StudentConnect 20](#_Toc106901879)

[3.1 Introducere 20](#_Toc106901880)

[3.2 Cerințe funcționale 20](#_Toc106901881)

[3.2.1 Cerințe funcționale ale aplicației Android 20](#_Toc106901882)

[3.2.2 Cerinte funcționale ale serverului Java 21](#_Toc106901883)

[3.3 Arhitectura sistemului 21](#_Toc106901884)

[3.4 Aplicația de tip client Android 22](#_Toc106901885)

[3.4.1 Descrierea aplicației 22](#_Toc106901886)

[3.4.2 Funcționalitățile aplicației 22](#_Toc106901887)

[3.4.2.1 Pagina de start 23](#_Toc106901888)

[3.4.2.1.2 Autentificare prin intermediul microserviciilor gRPC 24](#_Toc106901889)

[3.4.2.2 Pagina principală a aplicației 25](#_Toc106901890)

[3.4.2.3 Funcția de căutare 27](#_Toc106901891)

[3.4.2.4 Mesageria instant 27](#_Toc106901892)

[3.4.2.5 Pagina de profil 29](#_Toc106901893)

[3.4.2.6 Secțiunea de meniu 30](#_Toc106901894)

[3.4.2.7 Dependințe suplimentare 31](#_Toc106901895)

[3.5 Aplicația de tip server Java 32](#_Toc106901896)

[3.5.1 Descrierea aplicației 32](#_Toc106901897)

[3.5.2 Funcționalitatea aplicației de tip server 32](#_Toc106901898)

[3.5.2.1 Clasa principală a serverului 33](#_Toc106901899)

[3.5.2.2 Clase ajutătoare 33](#_Toc106901900)

[3.5.2.3 Servicii 33](#_Toc106901901)

[3.5.2.3.1 PostsService 34](#_Toc106901902)

[3.5.2.3.2 ImagesService 40](#_Toc106901903)

[3.5.2.3.3 LoginService 42](#_Toc106901904)

[3.5.2.3.4 ChatHistoryService 43](#_Toc106901905)

[3.5.2.3.5 SearchAndChatService 44](#_Toc106901906)

[3.5.2.3.6 MessagingService 49](#_Toc106901907)

[3.5.2.4 Clasa ChatSession 51](#_Toc106901908)

[3.6 Baza de date pentru aplicația de tip server 53](#_Toc106901909)

[4. Dezvoltarea si scalarea proiectului 55](#_Toc106901910)

[4.1 Dezvoltarea aplicației client Android 55](#_Toc106901911)

[4.2 Dezvoltarea aplicației server Java 56](#_Toc106901912)

[Concluzie 56](#_Toc106901913)

Bibliografie..............................................................................................................................................58

**1.Introducere**

În general, o rețea de socializare este un spațiu virtual, bazat pe noduri şi site-uri web, unde utilizatorii de internet cu aceleaşi interese se pot întâlni, cu scopul de a crea legături virtuale. Aceştia sunt conectați într-un mod informal şi fac schimb de informații.

Din punct de vedere istoric, rețelele de socializare sunt recente, apărând odată cu dezvoltarea tehnologiei care a oferit posibilitatea utilizatorilor de internet să trimită mesaje instant, să adauge comentarii pe forumuri şi să-şi facă liste de prieteni; adică din anul 1994. Prima rețea de socializare a fost numită „Sixdegrees.com”, unde aveai posibilitatea să îți creezi un profil personal cu o listă de prieteni şi să interacționezi cu ei sau să vizitezi profilurile acestora. Prima rețea de socializare din domeniul afacerilor a fost „Ryze.com” în timp ce prima rețea din domeniul întâlnirilor virtuale a fost „Friendster”. În anul 2003 a apărut bine-cunoscutul „Myspace”, iar în anul 2004 cea mai folosită şi populară rețea - „Facebook”, creată de Mark Zuckerberg.

De la prima rețea socială apărută în anul 1994, s-au creat nenumărate de rețele de socializare care se împart în 10 tipuri diferite:

1. Rețele sociale orizontale - create cu scopul de a facilita socializarea indivizilor cu preocupări şi interese diverse. Exemple: „Twitter”, „Facebook”.
2. Rețele sociale verticale - se mai numesc şi rețele sociale specializate şi se adresează unui anumit tip de public cu preocupări sau obiective asemănătoare.
3. Rețele sociale profesionale - unde utilizatorii sunt profesionişti şi acționează asupra unor obiective de lucru. Exemplu: „LinkedIn”.
4. Rețele sociale verticale mixte - sunt mai puțin formale deoarece combină temele de agrement cu cele profesionale.
5. Ştiri sociale - locul unde comunitatea de internet decide, prin sistem de vot, ce ştiri sunt relevante pentru public, postate de alți utilizatori sub formă de linkuri sau articole externe. Exemple: „Reddit”, „Digg”, „Meneame” (pentru utilizatorii de limba spaniolă).
6. Rețele sociale universitare- sunt destinate studenților din cadrul universitar, care le oferă posibilitatea de a discuta între ei şi de a afla informații utile. Exemplu: „Patatabrava”.
7. Rețele sociale de agrement- sunt rețele sociale verticale care au ca şi subiect sportul, muzica, jocurile video. Exemple: „Wimpley”, „Dogster”.
8. Blog-ul - se referă la un site dedicat publicării opiniilor, poveştilor, articolelor şi link-urilor sub formă de site personal, ce poate fi citit de orice vizitator. Exemple: „WordPress”, „Blogger”.
9. Microblogging – aceasta este o facilitate inclusă în majoritatea rețelelor de socializare, prin care se crează și se trimit hashtag-uri (etichete) pentru a accesa un anumit subiect, iar publicațiile sunt scurte. Exemple: „Twitter”, „Tumblr”.
10. Conținut partajat – este un spațiu online unde este permisă încărcarea şi partajarea de imagini şi conținuturi video. Exemple: „Youtube”, „Flickr”.

Având în vedere proporțiile uriaşe ale unei rețele de socializare, se creează în cadrul ei comunități care funcționează pe baza unor anumite tipologii de utilizatori, a unei structuri de organizare şi a unor reguli de utilizare stabilite de fondatorii rețelei.

1. Moderatorul - supraveghează conținutul postat, respectarea regulilor stabilite şi a scopurilor inițial declarate de către aderenții rețelei. Acest rol îi revine unei persoane sau unui grup de persoane.
2. Influențatorul - răspândeşte cel mai repede informația cu un număr mare de oameni deoarece este bine ancorat in lumea online și are un număr mare de contacte, fiind conectat cu receptori țintă din mai multe rețele și sisteme de interacțiune online.
3. Tăcutul - este factorul care nu îşi face simțită prezența activ, el doar vizitează şi consumă tacit conținutul postat și comentariile. În realitate el poate fi cel mai înfocat fan al comunității şi poate aduce membrii noi.
4. Agitatorul – este un membru indispensabil, care instigă mereu discuții contradictorii pe numeroase subiecte, ajutând la realizarea unei statistici asupra numarului de oameni care apreciază sau nu produsele sau serviciile.
5. Entuziastul - sau motorul comunității, este persoana care postează cel mai des, îşi exprimă cel mai mult opinia, ține comunitatea activă.
6. Curiosul - este si el o persoană importantă, mai ales datorită numărului mare de întrebări pe care le pune în mod frecvent.

Ca orice lucru, aceste site-uri prezintă avantaje şi dezavantaje, existând păreri împărțite referitoare la utilitatea lor. Cert e că, în pofida părerilor contradictorii, rețelele de socializare s-au consacrat în conştiința publica drept un canal facil de comunicare cu alți oameni; apariția şi utilizarea acestora fiind inevitabile.

Cel mai mare avantaj al rețelelor socializare este îmbunătățirea calității comunicării şi a schimbului de informații. Odată cu apariția posibilității de a trimite mesaje instant în toate colțurile lumii, au dispărut toate barierele de comunicare. Internetul şi rețelele de socializare facilitează foarte mult comunicarea interpersonală, astfel că dacă vrem să purtăm o conversație cu o persoană de oriunde de pe glob, tot ce trebuie să facem e să scriem un simplu mesaj şi să apăsăm butonul de send. Cealaltă persoană ne va răspunde aproape instant. E o îmbunatățire uriaşă care ne salvează foarte mult timp față de era în care trimiteam mesaje prin poştă sau alte prin alte căi şi răspunsul venea în câteva zile sau chiar mai mult.

Aceasta proprietate ne ajută nu numai în plan personal, dar și în cadrul spațiului de muncă. Astfel, putem rezolva multe dintre probleme legate de locul de muncă pot fi rezolvate fără a fi prezent în persoană, cu un simplu click. (Așa a apărut telemunca).

De asemenea, putem obține informații, fiind mereu conecțati cu ce se întămpla în lume și având acces nelimitat la orice tip de informație. În acest fel, beneficiem de educație continuă în orice domeniu dorim.

Timpul liber beneficiază de facilitățile oferite de rețelele sociale deoarece acestea pot oferi o gamă largă de informații și conținut: de la hobby-uri până la filme, jocuri, muzică, etc.

Pe partea opusă, site-urile ne vin în ajutor în momentul în care suntem în căutarea unui loc de muncă. Companiile folosesc platformele online și rețelele de socializare pentru a-și expune ofertele de muncă, deoarece acestea sunt un spațiu accesibil publicului larg.

Rețelele de socializare le oferă spațiu de promovare, unde primesc feed-back în timp real și își pot monitoriza imaginea publică, prin îmbunătățirea relației cu clienții și loializarea acestora.

Un rol foarte important pe care îl au platformele sociale este atragerea atenției către situații sociale de interes public.

Unul dintre dezavantajele majore ale utilizării rețelelor de socializare poate fi securitatea datelor personale. Probabil cu toții am întălnit sau am auzit de escrocheriile care au loc pe internet. Acestea se întâmplă atunci cand sistemul de securitate al rețelei de socializare utilizată nu asigură suficient protecția datelor confidențiale sau când utilizatorii ignoră instrucțiunile de securitate cu privire la distribuirea de informații confidențiale. În caz contrar aceștia ajung victimele escrocheriilor sau chiar a furturilor de identitate.

Un alt efect nedorit poate fi dependența de internet. Acest comportament este cel mai des întălnit la tineri, însă apare din ce în ce mai mult și la adulții sau varstnicii care în ultimii ani au devenit din ce în ce mai interesați de acest mod de divertisment. Peste 210 milioane de oameni din întreaga lume suferă de dependența de internet și de social-media. Persoanele dependente au ca și simptome depresie, anxietate, izolare socială, tendința de evitare a muncii, atitudine defensivă și multe altele.

Un dezavantaj important e fenomenul de ciberbullying si grooming unde țintele principale sunt copii și minorii din întreaga lume, astfel internetul reprezentând un spațiu periculos pentru aceste categorii de vârstă.

Mai exista și figura troll-ului, persoana care face critici și remarci cu scopul de a enerva și a instiga la ceartă pe rețelele de socializare.

Pentru companii, dezavantajele vin sub formă de recenzii nefavorabile și de strategii neproductive sau lipsite de creativitate.

Platformele de socializare pot avea efecte negative și asupra dezvoltării cognitive a copiilor și a adolescenților, provocându-le un grad ridicat de agresivitate.

1.2 Scopul

Ideea acestui proiect a plecat de la dorința de a găsi o soluție mai elegantă, mai ușoară si poate mai oficială pentru conectarea între studenți decât optiunile prezente, cum ar fi: grupuri de „Facebook” și/sau „WhatsApp”. Spre exemplu, odată cu începerea fiecărui nou an universitar, fiecare dintre noi trebuie să ceară permisiunea de a fi acceptat in diferite grupuri de cămin. Aceasta aplicație urmărește automatizarea procesul de atribuire a studențiilor în diferite categori, astfel experieța fiind transformată într-un proces fluid, insesizabil.

1.2.1 Obiective

Obiectivele generale dezvoltării unei aplicați sociale pentru studenții:

* Fluidizarea experienței de conectare
* Crearea unui spațiu de socializare
* Automatizarea atribuirilor prin uzul bazelor de date deja existente

**1.3 Comparația cu alte aplicații asemănatoare**

În momentul dezvoltării există multiple platforme si rețele de socializare care facilitează diferite moduri de conectare si socializare, însă acestea vin cu un efort în plus al utilizatorului pentru a se conecta cu oamenii despre care acesta este interesat. Aplicația propusă are următoarele avantaje:

* Adunarea tuturor studenților, si eventual al cadrelor universitare într-o singura platformă socială pentru ușurarea interacționării.
* Lipsa unui proces de înregistrare, facilitat de logarea prin intermediul adresei instituționale

Însă, aplicația propusă în aceasta lucrare de licență este la un nivel de prototip, având lipsă câteva aspecte atractive ale unei rețele sociale convenționale, cum ar fi:

* Trimiterea de poze în cadrul mesajelor private
* Distribuirea unei anumite postării spre o terță externă
* Utilizarea de etichete (hashtag-uri)

2. Tehnologiile software folosite

2.1 JetBrains IntelliJ IDEA

IntelliJ IDEA este un IDE conceput special pentru a scrierea în Java, Kotlin, Groovy și alte limbaje bazate pe JAR. Este dezvoltat de către JetBrains (înainte cunoscut ca și IntelliJ). Prima versiune de IntelliJ a fost lansată în Ianuarie 2021, și a fost unul dintre primele IDE-uri pentru Java, cu capabilități de navigare de cod avansată și refactorizare de cod, direct integrate în IDE.

Câteva dintre proprietățiile oferite de IntelliJ sunt:

* Completare de cod prin analizarea contextului
* Navigarea codului, care permite saltul la diferite clase sau linii de cod
* Refactorizarea codului
* Code debugging
* Opțiuni de rezolvare a inconsistențelor prin sugesti

Câteva dintre limbajele suportate de IntelliJ sunt:

* Groovy
* HTML, XML, JSON, YAML
* Java
* Kotlin
* JavaScrypt
* SQL
* PHP
* GO
* Ruby si JRuby

2.2 Programare în Java

Java este un limbaj de programare de nivel înalt, inițial create de Sun Microsystems in 1995. Este un limbaj de programare cu uz-general, proiectat să fie scris odată si rulat oriunde, pe multiple sisteme de operare cum ar fi Windows, MacOS si Linux, făra diferențe. Aplicațiile Java sunt de obicei compilate în bytecode, care poate fi rulat pe pe orice Java Virtual Machine (JVM). Java are o sintaxa asemănătoare cu C si C++. Limbajele compilate tradițional nu au capabilități dinamice (cum ar fi reflecția si modificare de cod la runtime), pe când Java ne pune la dispoziție aceste lucruri prin Java runtime.

Câteva dintre calitățile ale limbajul Java sunt:

* Programarea orientată pe obiecte.
* Indepența de platformă. La momentul compilării, acesta este transformat in bytecode, care poate fi rulat pe orice platformă, atât timp cât aceea platformă dispune de un JVM
* În continuarea punctului anterior, Java este un limbaj interpretat, bytecode-ul fiind translatat la momentul rulării în instrucțiunii specifice mașinii native.
* Java este un limbaj foarte dinamic
* Java a fost proiectat să fie begginer-friendly, astfel că este ușor de învățat
* Java suportă fire de execuție multiple
* Java este portabil si robust, din cauza neutralității de arhitectură si al efortului de a elimina situațiile în care pot apărea erori neprevăzute.
* Este un limbaj sigur, datorită caracteristicilor securizate ale limbajului

2.3 Android

2.3.1 Sistemul de operare Android

Este bazat pe o versiune modificată de nucleu Linux. Acesta a fost conceput in principal pentru dispozitive mobile cu ecran tactil, precum telefoane inteligente si tablete. Inițial dezvoltat de Google, în prezent de Open Handset Alliance. Cu timpul, Google a dezvoltat și alte sisteme de operare bazate pe Android cum ar fi: „Wear OS” pentru ceasuri inteligente, „Android TV” pentru televizoare inteligente și „Android Auto” pentru navigații de autoturisme. Android este cel mai vândut sistem de operare la nivel mondial, atingând peste două miliarde de utilizatori în anul 2017, iar magazinul său de aplicații având peste 3.48 milioane de aplicații.

Nucleul de Android este bazat pe nucleele LTS (long-term support) Linux. Google a implementat multe schimbării arhitecturale precum: arbori de dispozitive, ION, ashmem, OOM (out-of-memory) handling.

Fiecare aplicație Android rulează într-un așa numit „sandbox”, o zonă separată de restul sistemului, fără acces la acesta, decât cu acordul utilizatorului. Astfel, se crează o zonă securizată de rulare și aplicația are acces doar la resursele necesare rulării.

2.3.2 Dezvoltare de software în mediul Android

Dezvoltarea Android este un proces prin care se creează aplicați noi, cu ajutorul ”Software Development Kit”-ului de Android, pentru dispozitivele care rulează pe sistemul de operare Android.

2.3.2.1 Android SDK

Acesta reprezintă un set complet de instrumente de dezvoltare, incluzând: programe de depanare, biblioteci, un emulator de dispozitiv (bazat pe „QEMU”), documentație, exemple de cod și chiar tutoriale. Platformele care sprijină în prezent dezvoltarea Android sunt calculatoarele pe arhitectura x86 care rulează „Linux”, „MacOS X 10.4.8” sau mai recent, „Windows XP” sau mai recent. De asemenea, cerințele mai includ si „Java Development Kit” (JDK), „Apache Ant” și „Python 2.2” sau o versiune ulterioară.

În tabelul de mai jos putem observa istoricul versiunilor de Android, nivelul API-ului acestora, cât și ce procentaj din dispozitive rulează versiunea respectivă:

|  |  |  |  |
| --- | --- | --- | --- |
| **Versiune** | **Nume de cod** | **API Level** | **Distribuție** |
| 12 | Salad of Fruits | One UI 4.0,4.1 | 0.0% |
| 11 | Rhubarb Rolls | One UI 3.0, 3.1, 3.1.1 | 34.3% |
| 10 | Queen’s Cake | One UI 1.0-2.5 | 22.9% |
| 9.0 | Pie | 28 | 12.5% |
| 8.0-8.1 | Oreo | 26, 27 | 9.2% |
| 7.0-7.1 | Nougat | 24, 25 | 5.2% |
| 6.0 | Marshmallow | 23 | 2.7% |
| 5.0-5.1 | Lollipop | 21, 22 | 2.4% |
| 4.4 | KitKat | 19, 20 | 0.7% |
| 4.1-4.3 | JellyBean | 16, 17, 18 | 0.3% |

Codul aplicațiilor Android poate fi scris în limbaje precum Kotlin, Java sau C++, folosind desigur SDK Android.

2.3.2.2 „Android Studio IDE”

IDE-ul oficial pentru dezvoltarea aplicațiilor Android este „Android Studio”, dezvoltat de Google și bazat pe JetBrains IntelliJ IDEA, proiectat special pentru dezvoltarea software-ului Android. Acesta este disponibil pe Windows, MacOS si Linux.

Câteva dintre proprietățile prezente în ultima versiune de Android Studio sunt:

* Emulatorul Android Virtual Device pentru a rula și testa aplicațiile direct în Android Studio
* Support pentru compilare bazat pe Gradle
* Editorul IntelliJ IDEA
* Un editor de UI bogat, care permite construirea acestuia prin acțiuni de drag-and-drop
* Support pentru Google Cloud Platform, Firebase Cloud Messaging si Google App Engine

În final, aplicațiile sunt împachetate în format .apk, care servește drept instalator în mediul Android.

2.3.2.3 Componentele unei aplicați Android

2.3.2.3.1 Componentele fundamentale

1. **Activități**

O activitate este punctul de intrare în contact al aplicației cu utilizatorul. Aceasta reprezintă un singur ecran cu o interfață grafică. Deși activitățile lucrează împreună pentru a oferi o experiență continuă pentru utilizator, acestea pot lucra si exista independent una de cealaltă. O activitate oferă următoarele interacțiuni între sistem si aplicație:

* Ține cont de acțiunile utilizatorului pentru a asigura că sistemul ține in viață procesul care găzduiește activitatea
* Să țină minte ce procese au fost folosite înainte, și la care utilizatorul s-ar putea întoarce (spre exemplu, alte activități oprite), astfel prioritizând menținerea acestor procese
* Ajută aplicația cu terminarea proceselor, astfel încăt utilizatorul să se poate întoarce la activitățile precedente
* Permite o cale de distribuire a datelor între activități in cazul in care acest lucru este necesar

1. **Servici**

Un serviciu este un punct de intrare al aplicației pentru a menține aplicația în funcțiune în background, pentru diferite motive. De obicei, acesta este o componentă care efectuează operații lungi. Un serviciu nu oferă o interfață grafică. De exemplu, un serviciu ar putea să obțină date printr-o conexiune la internet, fără a bloca interacțiunea utilizatorului cu aplicația.

Exista două tipuri de servici:

1. „Started Services”, sunt servici pe care sistemul le menține pănâ când și-au terminat operațiunea pe care o aveau de făcut, oricare ar fi aceasta.
2. „Bound Services”, acestea ruleaza deoarece o altă aplicație sau chiar sistemul a pornit-o. Acestea sunt de obicei un serviciu care oferă un API altui proces. Astfel sistemul știe că există o dependință între procese.

**3. Broadcast Receivers**

Un „Broadcast Receiver” este o componentă care permite livrarea de evenimente către aplicație, înafara uzului normal al utilizatorului, permițând aplicației să răspundă la anunțuri de la sistem. Aceștia sunt alte puncte de intrare în aplicație, astfel că sistemul poate livra mesaje chiar și aplicațiilor care nu rulează la momentul livrări mesajului.

**4. ”Content Providers„**

Aceștia manageriază un set de date pe care aplicația le poate stoca într-un sistem de fișiere, fie într-o bază de date locală, fie pe internet, sau orice altă locație pe care aplicația o poate accesa.

Aceștia sunt foarte folositori pentru citirea și stocarea de date care sunt private, doar pentru aplicația respectivă.

2.3.2.3.2 Manifest file

Înainte ca sistemul Android sa poată să înceapă rularea unui component, acesta trebuie să știe dacă acesta există, citind fișierul AndroidManifest.xml. Aplicația trebuie să declare toate componentele în acest fișier, care trebuie să stea la baza directorului aplicației.

Acesta mai are si alte utilizări înafară de cele menționate mai sus, cum ar fi:

* Identificarea permisiunilor de care aplicația are nevoie că să ruleze, cum ar fi cea de Internet
* Să declare nivelul minim al API-ului pentru rularea aplicației
* Declararea caracteristicilor hardware si software pe care aplicația are nevoie să le folosească, cum ar fi accesul la camera, bluetooth etc.

2.3.2.3.3 Resursele aplicației

Resursele sunt fișiere suplimentare și conținut static pe care aplicația îl folosește, cum ar fi bitmaps, definiți de layout-uri, instrucțiuni pentru animații, etc.

Acestea ar trebui să fie exteriorizate codului aplicației, pentru a putea fi întreținute separat. Acestea pot fi accesate din interiorul codului printr-o serie de ID-uri de resurse care sunt generate in clasa **R** a proiectului.

2.3.2.4 Limbajul XML

XML este un limbaj de marcare si formatare de fișier pentru stocare, transmisie si reconstruire de date. Acesta definește o serie de reguli pentru codarea documentelor într-un format care să fie înțeles atât de către cititori umani, cât si cei virtuali. Dar aceste reguli sunt flexibile, ele pot fi create si modificate la latitudinea utilizatorului.

Scopul general al acestui limbaj este stocarea, transmiterea si reconstruirea de date definite arbritrar. Ca două sisteme diferite să interschimbe date, acestea trebuie să se convină asupra unui format de fișier. Aici intervine XML-ul, standardizând acest proces.

Acesta seamănă cu limbajul HTML, doar că este mai dinamic, diferența esențială este că tagurile nu sunt predefinite.

2.3.2.4.1 XML în Android Development

În dezvoltarea aplicațiilor Android sunt folosite fișiere de tip XML pentru definirea unor resurse denumite layout-uri. Aceste layout-uri definesc structura interfeței grafice a diferitor părți din aplicație, cu care utilizatorul poate interacționa direct.

Layout-uri sunt construite folosind o ierarhie de obiecte de tip „View” si „ViewGroup”. Un „View” de obicei desenează un obiect cu care utilizatorul poate interacționa, pe când „ViewGroup” este un container invizibil ce definește structura din layout al View-ului.

Android ne pune la dispoziție un întreg vocabular XML pentru a putea proiecta layout-uri interfețelor grafice, și pentru elementele din înauntrul acestora. Fiecare fișier de layout trebuie să conțină exact un element părinte, după care se pot face diverse combinații ierarhice după plăcerea și idea programatorului. Acestea trebuiesc neapărat salvate in folderul „res/layout/„ din proiectul Android pentru a putea fi folosite ca atare.

ViewGroup oferă o multitudine de modalității de afișare a conținutului. Unele dintre cele mai folosite sunt:

* „ConstraintLayout”
* „LinearLayout”
* „GridLayout”
* „RelativeLayout”

Toate View-urile au seturi de atribute care modifică modul în care acestea sunt afișate pe ecranul telefonului. Unele atribute sunt comune pentru toate View-uri, iar câteva dintre acestea sunt:

1. „id” = un ID unic care corespunde view-ului si este folosit pentru identificarea acestuia la runtime
2. „layout\_width” = lățimea view-ului ( obligatoriu pentru orice view )
3. „layout\_height” = înalțimea view-ului ( obligatoriu pentru orice view )
4. „layout\_margin” = extra spațiu înconjurul view-ului
5. „layout\_marginTop” = extra spațiu deasupra view-ului
6. „layout\_marginBottom” = extra spațiu sub view
7. „layout\_marginStart” = extra spațiu în stângă view-ului
8. „layout\_marginEnd” = extra spațiu în dreapta view-ului
9. „padding” = bordură înconjurul view-ului
10. „paddingTop” = bordură deasupra view-ului
11. „paddingBottom” = bordură sub view
12. „paddingStart” = bordură în stânga view-ului
13. „paddingEnd” = bordura în dreapta view-ului

2.4 „SQL”

„SQL” este un limbaj de programare specific pentru manipularea datelor într-un sistem de gestiune a bazelor de date relaționale, iar originea acestui limbaj este algebra relațională. Acesta are ca și scop inserarea, actualizarea și ștergerea datelor, modificarea și crearea schemelor relaționale, precum și controlul accesului la aceste date. Acesta a devenit cel mai popular limbaj utilizat pentru astfel de scopuri din domeniul, fiind cel mai folosit de catre SGBD-uri. De asemnea există și o mulțime de dialecte si variante ale acestuia, unele conținând extensii pentru a suporta BSD-uri

(Sisteme de Baze de Date) obiectuale(obiectual-relaționale).„SQL” permite accesul la conținutul bazelor de date, cât si la structura acestora.

„SQL” a fost inițial dezvoltat in anul 1974, în cadrul companiei „IBM” de către Donald D. Chamberlin si Raymond F. Boyce, și inițial a avut ca și scop manipularea și regăsirea datelor stocate în bazele de date.

Oricăre câmp sau coloană dintr-un tabel SQL are atribuită un tip de dată, ca în orice alt limbaj de programare. Tipuri de date sunt:

* „CHARACTER” (sau CHAR) – pentru șiruri de caractere
* „INTEGER” – pentru numere întregi
* „FLOAT, REAL SAU DOUBLE PRECISION” – pentru numere reale
* „NUMERIC(precision, scale) ” sau „DECIMAL(precision, scale) ” – pentru numere zecimale, unde “precision” înseamnă numărul de cifre din partea întreagă, și „scale” înseamnă numărul de zecimale
* „DATE” – pentru data zilei
* „TIME” – pentru oră
* „NOW” – pentru data si ora curentă

Clauze de interogare:

* „SELECT” – pentru selecția datelor
* „INTO” – pentru a transfera selecția într-o tabelă nouă
* „WHERE” – pentru specificarea condițiilor care trebuiesc împlinite pentru ca rândul de date să fie luat în considerare

Instrucțiuni de manipulare a datelor:

* „INSERT” – pentru inserarea datelor noi într-un tabel
* „UPDATE” – pentru actualizarea datelor dintr-un tabel
* „DELETE” – pentru a șterge date dintr-un tabel

Limbajul de definire a datelor/tabelelor:

* „CREATE TABLE” – pentru crearea unui nou tabel
* „ALTER TABLE” – pentru modificarea unui tabel
* „DROP TABLE” – pentru stergerea totală a unui tabel

**2.4.1 H2 Java Database Engine**

H2 este un Sistem de gestionare a bazelor de date, scris în Java. Acesta poate fi încorporat în aplicați Java pentru gestionarea bazei de date direct din aplicație. Este disponibil ca și software open-source sub Licența publica Mozilla 2.0.

Acesta permite accesul ușor la daza de date din interiorul aplicației pentru manipularea acesteia cu ușurinta printr-o biblioteca proprietara, și de asemenea un client web prin care se poate interacționa direct cu baza de date, fără a mai fi nevoie de programe terțiare. De asemenea, aceasta este foarte rapidă în utilizare, bazele de date sunt cryptate, accesul la aceasta se face prin roluri si parole criptate.

2.5 gRPC

gRPC (g – Remote Procedure Calls) este un framework modern cu performanță ridicată de Remote Procedure Call, care poate rula în orice mediu. Acesta a fost dezvoltat inițial de Google, care a folosit o infrastructură general-purpose RPC și a numit-o ”Stubby”, pentru a conecta un număr mare de microservici care rulau în interiorul centrelor de date. În Martie 2015, Google a decis să construiască următoare versiune de Stubby și să o facă open-source. Rezultatul a fost gRPC, care acum a fost adoptat de multe organizații din lume pentru a folosii infrastructura bazată pe microservici. Acesta folosește HTTP/2 pentru transportul datelor și ”Protocol Buffers” ca și limbaj descriptiv de interfață.

gRPC permite servici precum:

* Autentificare
* Stream bidirecțional de date
* Flow control
* Cancellation
* Timeouts

Acesta generează clienți si server, independenți de platforma, pentru mai multe limbaje de programare. Una dintre cele mai folosite paradigme de utilizare a acestui framework este conectarea dispozitivelor mobile la servicii de tip backend.

O mulțime de aplicații mobile folosesc acest framework pentru comunicare, cum ar fi:

* Uber
* Netflix
* Spotify

2.5.1 Protocol Buffers

Este un format gratuit si open-source cross-platform, folosit să serializeze date structurate. Este folositor în dezvoltarea de programe care comunică prin intermediul unei rețele sau care stocheaza informații. Metoda implică structurarea unor date si generarea unui cod sursă care generează sau analizează un flux de octeți care reprezintă datele structurate anterior.

Google a dezvoltat Protocol Buffers pentru uz intern și a furnizat un generator de cod sursa pentru mai multe limbaje, sub o licență open-source. Scopul dezvoltării Protocol Buffers a fost simplitatea si performanța. În particular a fost proiectat să fie mai mic si mai rapid decât XML-ul. Acesta este folosit foarte mult în interiorul Google pentru a stoca si a schimba tot felul de date structurate. Aceasta metodă folosețe drept bază pentru un sistem RPC personalizat, care este folosit în toate comunicările de tip mașina-mașina la Google.

Structurile de date, care sunt numite mesaje, si servicile sunt descrise in fișiere .proto si compilate cu „protoc”. Compilarea generează cod care poate fi invocat fie de transmițătorul, fie de receptorul acestor date structurate. Mesajele sunt serializate în fluxuri binare, care sunt compacte, și compatibile în ambele sensuri ale transmiterii.

Scopul principal al Protocol Buffers este de a facilita comunicare prin rețea, prin simplicitate si viteză.

Exemplu de fisier Protocol Buffers care descrie datele si serviciul de logare al aplicației:



3. Aplicația StudentConnect

3.1 Introducere

Această aplicație a pornit de la idea creări unei unelte oficiale de socializare și conectare între studenți, un spațiu de comunicare si împărtașire de informații între colegi, fără a mai fi nevoie de un efort din partea fiecărui student pentru a putea ajunge în interiorul acestui spațiu, toată atribuirea de date, conturi și rețele fiind automatizată de catre un sistem centralizat.

**Această platformă de socializare este compusa din 2 părți:**

* Aplicația de tip client, unde utilizatorii vor putea interacționa
* Serverul centralizat, care se v-a ocupa de distribuirea datelor, către clienți

Aplicația Android reprezinta interfața grafică, prin care studenți vor putea interacționa. Aplicația trimite date prin intermediul microserviciilor gRPC către server, care mai apoi propagă datele către restul utilizatorilor, pentru consum.

Serverul reprezinta partea de backend a aplicației, care va primi date prin intermediul microserviciilor gRPC, le va stoca în baza de date, pentru a fi folosite la momentul oportun, va trimite răspunsuri clienților, va regăsi date stocate si le v-a propaga, va primi, stoca si trimite poze înapoi la clienți.

3.2 Cerințe funcționale

3.2.1 Cerințe funcționale ale aplicației Android

* Logarea utilizatorului în aplicație
* Primirea si transmiterea de date spre și de la server
* Afișarea datelor mai sus menționate

Aplicația de tip client pe platforma Android reprezintă interfața grafică interactivă, prin care se va facilita comunicarea și socializarea între studenți. Acesta va acționa ca și o terță între utilizatorul real și server pentru a asigura validarea datelor, în cazul în care utilizatorii introduc date eronate, care ar putea duce la comportamente incorecte ale sistemului. Aceasta este concepută pentru:

* eficiență in rulaj, sistemul funcționând cu rapiditate, fiind constrâns doar de viteza internetului
* utilizabilitate, având o interfață prietenoasă si intuitivă, fiind foarte ușor de folosit
* compatibilitate, posibilitatea de rulare pe cât mai multe versiuni de Android
* siguranță în trasmiterea datelor, sistemul având mereu nevoie de un răspuns de la server, pentru a se asigură ca datele expuse sunt reale si corecte

3.2.2 Cerinte funcționale ale serverului Java

* Facilitarea logării în aplicație
* Primirea și transmiterea de date spre și de la client
* Stocarea și regăsirea datelor mai sus menționate în baza de date

Aplicația server reprezintă partea de backend a platformei noastre. Ea se ocupă de primirea datelor din partea clienților, de stocarea acestora în baza de date, propagarea acestora către clienți potriviți la momentul interogări, și de asigurarea securități datelor și a corectidudini acestora. Aceasta a fost concepută pentru asigurarea următoarelor calități:

* Eficiența în rulaj, de la microserviciile care rulează asincron pe sistem, pentru a permite satisfacerea mai multor interogării simultan, la sistemul de gestiune al bazei de date H2, care lucrează foarte eficient pentru preluarea datelor din interiorul bazei de date.
* Compatibilitatea, acesta fiind scris în Java, serverul poate rula pe orice sistem care suportă execuția de programe tip .jar, atâta timp cat sistemul respectiv dispune de o mașina virtuală Java
* Modificarea și întreținerea bazei de date prin ușurința de utilizare a clientului web H2 pus la dispoziția noastra, pentru a nu avea nevoie de programe terțiare pentru acest lucru, și ușurința acestui lucru prin interfața prietenoasa a acestuia pentru un tehnician IT.

3.3 Arhitectura sistemului

Sistemul este împarțit în aplicația mobilă de tip client Android, care acționează ca interfață grafică pentru utilizatori, și serverul scris în Java pentru asigurarea conexiunii dintre utilizatori și prelucrarea, stocarea și propagarea datelor la momentele interogării serverului. Datele sunt transmise în timp real de la server la client, transmisia datelor fiind începută de către o cerere din partea clientului, printr-un microserviciu gRPC pentru un anumite tip și set de date necesare afișării. Sistemul este special conceput pentru a rula și a permite conectivitatea mai multor dispozitive, simultan.

3.4 Aplicația de tip client Android

3.4.1 Descrierea aplicației

Aplicația de tip client Android pe nume StudentConnect reprezintă interfața grafică prin care utilizatori platformei vor interacționa și socializa între ei. Scopul aplicației este crearea spațiului de interacțiune dintre studenți, prin facilitarea unei platforme sociale, cu o interfața user-friendly. De asemenea, aplicația este un punct de intrare in sistemul informațional central al platformei, aceasta comunicând constant cu serverul pentru transmiterea, stocarea si propagarea datelor si mesajelor, în timp real.

3.4.2 Funcționalitățile aplicației

Aplicația este concepută pentru a facilita accesul constant și instant la date, mesaje și evenimente, ce sunt propagate prin intermediul sistemul informațional.

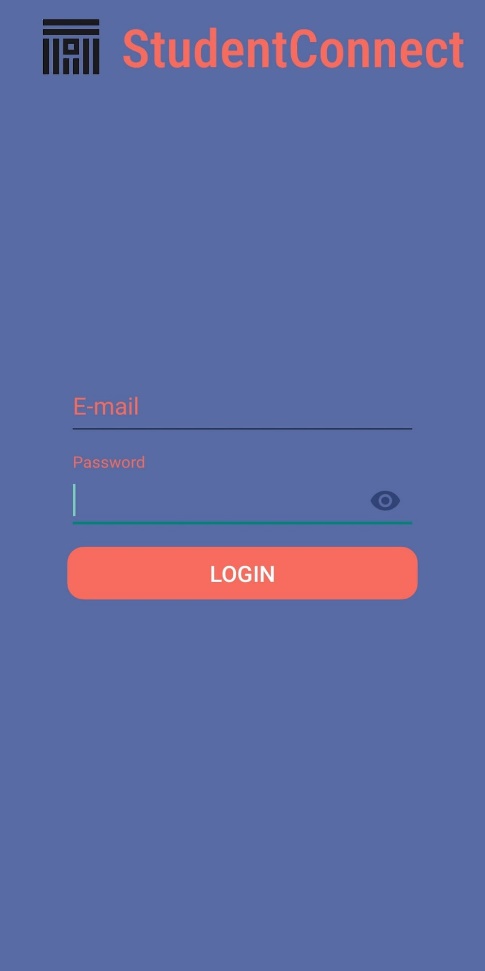
În prima fază, utilizatorul este nevoit să se autentifice în sistem prin intermediul adresei de e-mail instituțională, dupa care, aplicația va lansa pagina principală a aplicației, unde se pot vizualiza postările distribuite pe platformă, grupate în categorii, cum ar fi: pe cămine sau pe complexe studențești. Tot din aceasta pagina utilizatori pot crea postării noi pentru distribuirea acestora în sistem. De asemenea tot din aceasta pagina, utilizatori dispun de o funcție de căutare a persoanelor care fac parte din ecosistemul universitar, de unde le pot vedea profilele universitare, toate postările distribuite vreodată și chiar să inițieze o sesiune de mesagerie instant cu aceștia.

Tot din pagina principală a aplicației, utilizatori pot accesa istoricul conversațiilor avute prin intermediul serviciului de mesagerie instant, și pot continua conversații deja începute. Toate mesajele sunt stocate în interiorul bazei de date din server, astfel că acestea nu vor fi pierdute, decât în circumstanța în care mesajul nu a ajuns la server din cauza unei pierderi de conexiune a clientului.

3.4.2.1 Pagina de start

Pagina de start reprezinta defapt pagina de autenficare, unde utilizatorul completează câmpurile de e-mail si parolă, și la apăsarea butonului de login, aplicația trimite catre server o cerere de autentificare. Serverul răspunde în unul din două moduri:

* Contul exista și este corect, iar serverul permite accesul în aplicație, trecându-se la ecranul principal al aplicației cu datele utilizatorului logat
* Contul nu există sau este incorect, fapt ce determină serverul să nu permita accesul în aplicație



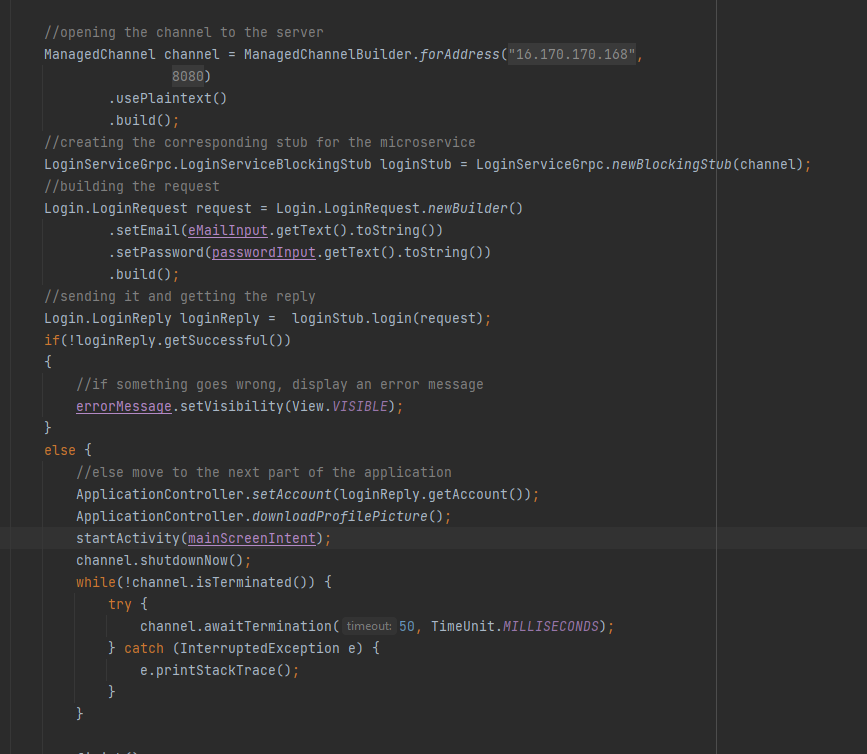
(Imagine cu pagina de autentificare a aplicației)

3.4.2.1.2 Autentificare prin intermediul microserviciilor gRPC

Odată cu apăsarea butonul de Login, clientul trimite o cerere de logare către server, care acesta, dupa procesarea cereri, va determina daca contul este sau nu unul valid, și va permite eventuala logare în aplicație, permițând trecerea la ecranul următor, pagina principală a aplicației.

Cererea de logare se crează în interiorul activității de logare astfel: odată cu apăsarea butonului de login se deschide un canal de comunicare către adresa serverului, cu portul pe care acesta rulează. Se crează un stub de gRPC special pentru microserviciul de logare. Se crează o instanță de obiect de tipul ”LoginRequest” care conține datele pentru acțiunea de login, și aceasta se populează cu datele introduse de utilizator. Se trimite request-ul cu ajutorul stub-ului către server, și se asteaptă răspunsul acestuia, pentru a continua rularea programului. În cazul în care contul este unul eronat, aplicația va afișa un mesaj de eroare informațional pentru utilizator, prin care îl va avertiza asupra faptului ca datele introduse de acesta sunt ori eronate, ori nu exista în sistem contul respectiv. Dacă, în schimb, răspunsul este unul favorabil, aplicația memoreaza datele despre utilizatorul curent primite de la server, și începe derularea următoarei activăți, cea de pagină principală.

În poza de mai jos putem observa codul descris:



3.4.2.2 Pagina principală a aplicației

Pagina principală a aplicației este compusă în principal de un ViewGrup de tipul RecyclerView, unde ne folosim de clasa ”PostsAdapter”, aceasta fiind o clasa care derivează clasa RecyclerView.Adapter, pentru a încarca și afișa postările distribuite pe platformă, în categoriile aferente. Fiecare celula din interiorul aceastei componente este o postare individuală, care poate avea comentarii de la orice utilizator și utilizatorii își pot arăta aprecierea, apăsând butonul specific pentru această acțiune.

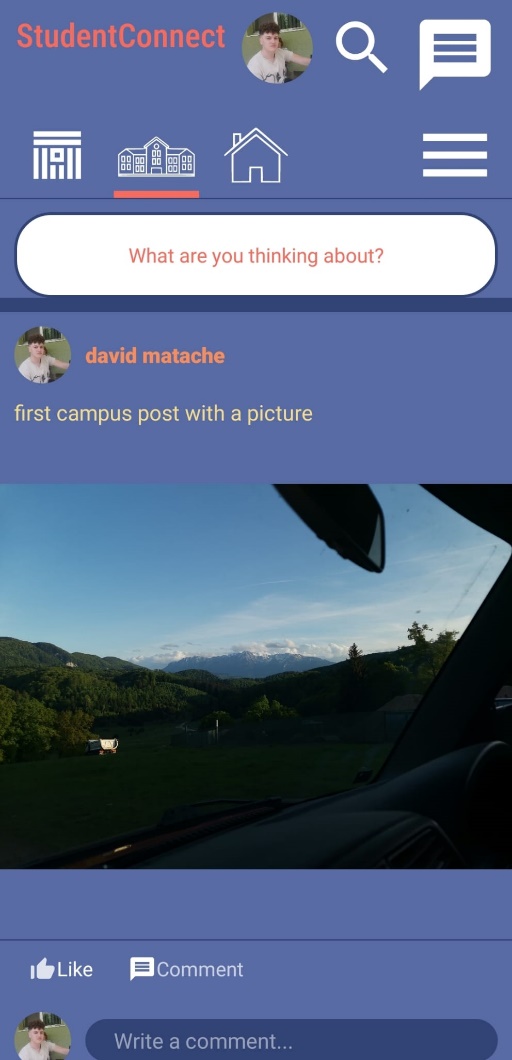
Aici, găsim și butonul pentru accesul serviciul de mesagerie, vizualizarea doar anumitor postării, și secțiunea de creare și distribuire a unei postări noi.

La momentul începeri activități de pagina principală se trimite către server un obiect de tipul ”PostRequest” pentru procurarea postărilor de pe server, clientul așteptă răspunsul serverului, după care încarcă diferitele postări cu ajutorul clasei PostsAdapter, în ViewGrupul de tipul RecylerView pentru afișarea elementelor grafice. Aceste postări pot avea si poze atașate.

De asemenea, deși nu este obligatoriu, fiecare profil are și o poza de profil. Aceste imagini se procura separat de postări dupa server, printr-un microserviciu dedicat transferului de poze. Aceste poze pot avea dimensiuni mari, astfel că primirea lor de la server se face cu ajutorul Thread-urilor, pentru a fermite fluiditate în interiorul aplicației. Odată ce una dintre poze a fost încărcată complet, aceasta este afișată în locul corespunzător, neîntrerupând acțiunile utilizatorului din aplicație. Fără aceste Thread-uri separate, utilizatorul ar trebui să aștepte încărcarea pozelor înainte de continuarea uzului aplicației, un lucru foarte neplăcut. Pentru a menține codul curat și ușor de citit, această încărcare a imaginilor se face în interiorul clasei PostsAdapter, pentru fiecare postare în parte, astfel că, acolo unde nu avem nevoie să încărcam vreo imagine, fie ea parte dintr-o postare, sau o poză de profil. Ne trebuie de asemenea și un obiect de tipul ”Handler” pentru a putea efectua actualizării grafice din interiorul Thread-urilor secundare.



(Imagine cu funcția de încărcare a unei imagi de pe server)

****

(Imagine cu ecranul principal al aplicației, care afișează una dintre postările disponibile)

3.4.2.3 Funcția de căutare

În partea de sus a ecranului, găsim o pictogramă care reprezintă o lupă. Apăsând aceasta imagine, putem accesa funcția de cautare a aplicației. Aici putem căuta după nume, membrii ai platformei. În momentul în care am găsit persoana căutată, putem să inițiem o conversație cu acestia, prin apasarea butonul de începere unei conversați, sau putem să vizualizăm întreg profilul utilizatorului respectiv, unde putem să vizualizăm toate postările făcute vreodată de acest utilizator, să interacționăm cu aceste postări, și să îi vizualizăm mai de aproape poza de profil, dacă utilizatorul dispune de una.

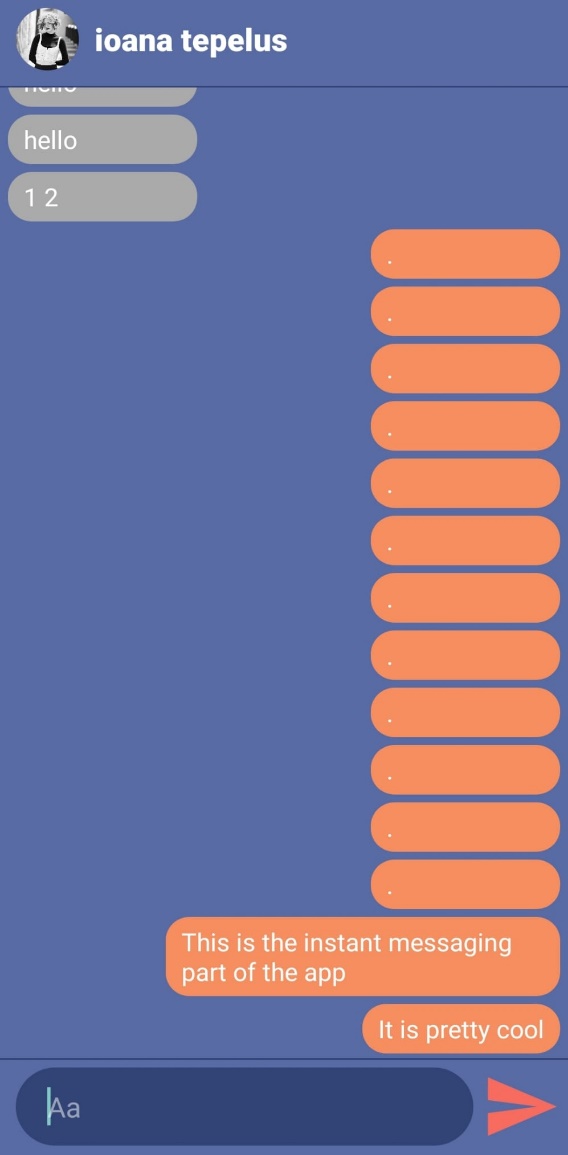
Activitatea de cautare este concepută dintr-o bară de căutare, unde putem introdu-ce indicii de căutare pentru găsirea utilizatorul dorit, iar mai apoi rezultatele acestei căutarii, prezentate într-un ViewGroup de tipul RecyclerView. Fiecare rezultat al căutari este afișat într-o celulă de RecyclerView. Pentru atașarea informațiilor în aceste celule ne folosim de clasa ”SearchAdapter” care moșteneste clasa ”RecyclerView.Adapter”. Odată cu interacționarea cu una dintre rezultatele căutării, aplicația va schimba activitatea către una nouă, fie ea de vizualizare a profilului sau de mesagerie instant.



3.4.2.4 Mesageria instant

Funcția de mesagerie instant permite comunicarea în timp real cu un alt utilizator al aplicației prin intermediul interfeței grafice. Aceasta activitate este concepută în principal dintr-un ViewGroup de tipul RecyclerView, unde sunt stocate, afișate și adăugate în timp real mesajele transmise între cei doi utilizatori. Deasupra acestuia, găsim poza de profil a utilizatorului cu care conversăm și numele său. În josul ecranului găsim secțiunea de introducere și transmitere a mesajului nostru. Aici avem o bară de editare text în care putem introduce mesajul dorit, și în dreapta acesteia regăsim butonul de trimitere al mesajului.

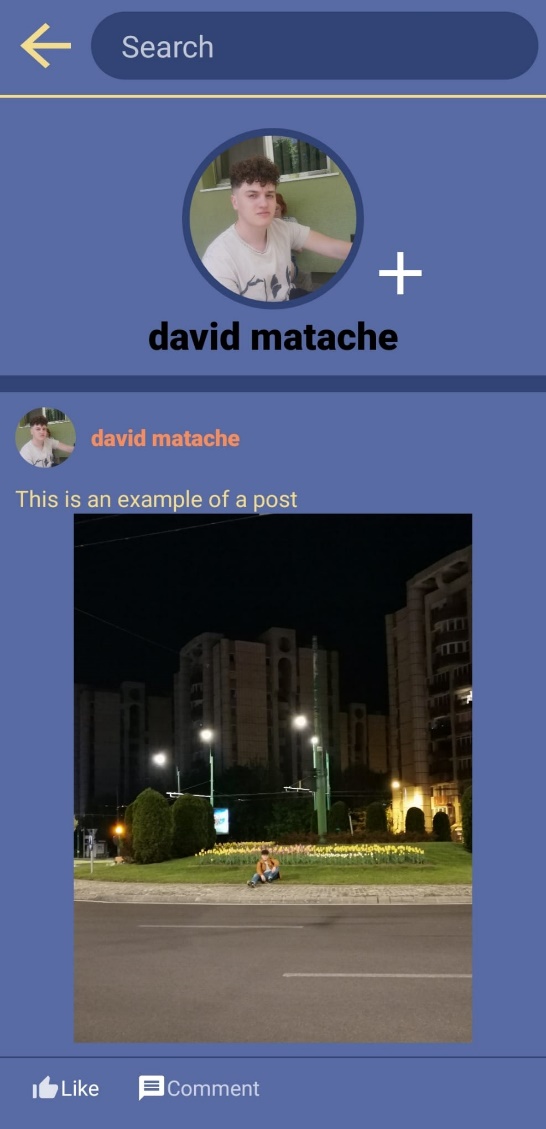
Atunci când activitatea de mesagerie este începută, aceasteia îi este transmis o valoare numerică, care reprezinta ID-ul sesiuni de mesagerie instant în cadrul serverului. Mai întâi, activitatea recuprează mesaje anterior trimise, pentru crearea istoricului de mesagerie, dacă există. După care, deschide canalul către server și inițiază conexiunea live către acesta prin request-ul de ”sendReceiveMessage”, astfel de fiecare dată cand serverul primește un mesaj pentru aceasta conversație, el ne este trimis imediat pentru afișare. La fel și atunci când noi trimite un mesaj, serverul îl primește și îl expediază mai departe către recipientul dorit.



(Imagine cu o sesiune de mesagerie instant)

3.4.2.5 Pagina de profil

Pagina de vizualizare a unui profil de utilizator este locul unde putem să găsim toate postările create si distribuite vreodată de un utilizator în cadrul platformei de socializare. Aceasta este concepută dintr-o bară de căutare, de unde putem să accesăm activitatea de căutare menționată anterior. Aici mai regăsim și poza de profil a utilizatorului într-o dimensiune mai mare. Tot restul paginii este alcătuit dintr-un ViewGroup de tipul RecylerView unde se află în ordine cronologică toate postările distribuite vreodată pe platformă, de la cea mai nouă la cea mai veche postare a utilizatorului respectiv. Postările sunt încărcate de pe server prin aceleași modalități ca și în interiorul activități de pagină principală, prin intermediul unui request, deși de data aceasta, specializat și concentrat doar pe un singur utilizator, unde răspunsul la aceasta cerere vor fi toate postările utilizatorului respectiv. Și aici, ca și în pagina principală a aplicației, ne folosim de clasa PostsAdapter pentru a crea si popula celule interfeței grafice cu postările mai sus menționate.



(Imagine cu pagina de profil a unui utilizator)

3.4.2.6 Secțiunea de meniu

Secțiunea de meniu reprezinta o activitate scurtă, de unde putem face un număr limitat de lucruri. Cel mai important lucru accesibil de aici de funcția de delogare a aplicației prin care putem să ieșim din sesiunea curentă care este conectată la un anumit cont, și să începem o sesiune nouă, posibil cu alt cont. Ne putem vizualiza poza de profil, putem încarca o poza nouă, lucru pe care îl putem realiza și accesandu-ne propria pagina de profil, mentionată anterior, sau ne putem accesa profilul personal.

De menționat ar fi faptul că, apăsând pe orice poză de profil regăsită în cadrul aplicației, vom accesa pagina de profil a utilizatorului respectiv.



(Imagine cu pagina de meniu a aplicației)

3.4.2.7 Dependințe suplimentare

* Pentru folosirea framework-ului de gRPC avem nevoie de câteva dependințe suplimentare în interiorul proiectului nostru de Android Studio, în fișierul de build.gradle al modulului aplicației
  + „implementation ‘io.grpc:grpc-okhttp:1.35.0’ ”
  + „implementation ‘io.grpc:grpc-protobuf-lite:1.47.0’ ”
  + „implementation ‘io.grpc:grpc-stub:1.47.0’ ”
* Pentru folosirea ViewGroup-ului de tipul RecyclerView este nevoie de:
  + „implementation ‘androidx.recyclerview:recyclerview:1.2.1’„
* Pentru realizarea interfeței grafice a pozelor de profil utilizate în aplicație este nevoie de:
  + ”implementation ‘androidx.appcompat:appcompat:1.6.0-alpha05’ ”
  + „implementation ‘de.hdodenhof:circleimageview:3.1.0’”
  + „implementation ‘androidx.cardview:cardview:1.0.0’”
* De asemenea, pentru generarea corectă a claselor de către framework-ul gRPC, este nevoie, în același fișier de:
  + În tagul ”plugins” este nevoie de:
    - id ‚’com.google.protobuf’
  + Următorul script pentru protobuf:



3.5 Aplicația de tip server Java

3.5.1 Descrierea aplicației

Aplicația de tip server Java este resursa de backend a platformei noastre, care se ocupă de:

* Asigurarea conexiuni între diferiți utilizatori ai platformei
* Stocarea datelor și informațiilor
* Regăsirea datelor din interiorul bazei de date
* Trimiterea, la cerere, a acestor date, înapoi la utilizatori, prin intermediului microserviciilor
* Asigurarea corectitudini și siguranței datelor menționate

3.5.2 Funcționalitatea aplicației de tip server

Aplicația de tip server este proiectată să asculte pe canalele de microservici pentru request-uri din partea diferiților clienți, și să răspundă la aceste cereri de date cu rapiditate si eficiență. Pentru fiecare microserviciu chemat, există modificări, inserări și regăsire de date aferente. Serverul trebuie să funcționeze constant pentru a asigura un serviciu fluid. Prin uzul microservicilor, serverul este mai ușor de citit de către programatori, și ușor de modificat și scalat la dimensiuni mai mari, pentru nivele mai mari de utilizatori. Acesta se folosește de sistemul de gestiune de baze de date H2, pentru rapiditatea în stocare și regăsire a datelor în baza de date aferentă.

3.5.2.1 Clasa principală a serverului

În interiorul clasei principale a aplicației server, se apeleaza funcția main de plecare din dialectul Java, unde se construiește serverul definit de pachetul „io.grpc.Server”, și la momentul creări se adăugă și toate microserviciile necesare rulării platformei noastre sociale, care sunt:

* + ChatHistoryService – pentru recuperarea sesiunilor de chat avute
  + ImagesService – pentru transportul imaginilor, în ambele sensuri de flux
  + LoginService – pentru autentificarea în platforma
  + MessagingService – pentru mesageria instantă a platformei
  + PostsService – pentru distribuirea și regăsirea postărilor din cadrul platformei
  + SearchAndChatService – pentru funcția de căutare de persone, și pentru eventuala începere a unei noi sesiuni de mesagerie instant, pornită din acest context de căutare.

3.5.2.2 Clase ajutătoare

În proiect găsim interfața ImageSaving, care declară funcția de salvare a unei imagini pe server, pornind de la un flux de biți și informații despre imagine.

Aceasta este implementată în clasa ImageActions, unde funcția este suprascrisă pentru salvarea propriu-zisă. Această clasă este folosită în cadrul serviciului ImagesService.

3.5.2.3 Servicii

Serviciile sunt miezul aplicației noastre server, care facilitează conexiunea și interacționarea cu instanțele de clienți. Acestea preiau cererile de la clienți și răspund în concordanță cu acestea. În aplicația noastră avem șase servicii cu diferite funcții și întrebuințări, așa cum urmează să fie prezentate în cele următoare:

3.5.2.3.1 PostsService

Serviciul PostsService se ocupă de toate funcțiile și acțiunile care au legătura cu postările din interiorul platformei. Aceste primește noi postări pentru stocare și propagare, memorându-le în interiorul bazei de date. Acest serviciu răspunde cerințelor clienților de propagare a postărilor, regâsind diferite postări, care respectă cerințele date de client, și le propagă înapoi la acesta.

Acest serviciu are în interiorul lui 6 microservicii, care se ocupă de diferite funcții legate de postări:

* + Microserviciul ”getPosts” – acest microserviciu regăsește în baza de date postării conform cu cerința clientului, și le propagă înapoi la client. În interiorul ei, se construiește o interogare SQL conform cerinței clientului. După, pentru fiecare postare găsită, se produce o nouă interogare, pentru a afla dacă clientul respectiv a apreciat postarea. În final rezultatele interogării se împachetează în tipul de dată Post și se trimite înapoi la client. Mai jos regăsim codul sursă pentru acest microserviciu:
* @Override  
  public void getPosts(Posts.PostRequest request, StreamObserver<Posts.Post> responseObserver) {  
   //Get posts from database and send them to the client  
   Statement statement = null;  
   try {  
   Connection con = DriverManager.*getConnection*("jdbc:h2:~/Licenta", "root","");  
   statement = con  
   .createStatement();  
   String sql;  
   String select;  
   if(request.getLastPostFetchedId() > 0) {  
   select = *selectSql* + *andPostID*;  
    
   if(request.getPostCategory() == Posts.PostCategory.*POST\_CATEGORY\_CAMPUS*){  
   sql = String.*format*(select + *campus* + *orderBy*,  
   parsePostCategoryToString(request.getPostCategory()),  
   request.getLastPostFetchedId(),  
   parseCampusToString(request.getWhichCampus()));  
   }  
   else if(request.getPostCategory() == Posts.PostCategory.*POST\_CATEGORY\_HOUSE*){  
   sql = String.*format*(select + *house* + *orderBy*,  
   parsePostCategoryToString(request.getPostCategory()),  
   request.getLastPostFetchedId(),  
   request.getWhichHouse());  
   }  
   else {  
   sql = String.*format*(select + *orderBy*,  
   parsePostCategoryToString(request.getPostCategory()),  
   request.getLastPostFetchedId());  
   }  
   }  
   else {  
   select = *selectSql*;  
   if(request.getPostCategory() == Posts.PostCategory.*POST\_CATEGORY\_CAMPUS*){  
   sql = String.*format*(select + *campus* + *orderBy*,  
   parsePostCategoryToString(request.getPostCategory()),  
   parseCampusToString(request.getWhichCampus()));  
   }  
   else if(request.getPostCategory() == Posts.PostCategory.*POST\_CATEGORY\_HOUSE*){  
   sql = String.*format*(select + *house* + *orderBy*,  
   parsePostCategoryToString(request.getPostCategory()),  
   request.getWhichHouse());  
   }  
   else {  
   sql = String.*format*(select + *orderBy*,  
   parsePostCategoryToString(request.getPostCategory()));  
   }  
   }  
   ResultSet resultSet = statement.executeQuery(sql);  
   while(resultSet.next())  
   {  
   statement = con.createStatement();  
   ResultSet liked = statement.executeQuery(String.*format*("SELECT liked from Likes WHERE POST\_ID = '%s' AND ACCOUNT\_ID = '%s'",  
   resultSet.getInt("POST\_ID"),resultSet.getInt("POSTER\_ID")));  
   boolean isLiked = false;  
   if(liked.next())  
   isLiked = (liked.getInt("liked") == 1);  
    
   boolean hasPhoto = resultSet.getInt("HAS\_PHOTO") == 1;  
    
   Posts.Post post = Posts.Post.*newBuilder*()  
   .setPostId(resultSet.getInt("POST\_ID"))  
   .setLitePost(Posts.LitePost.*newBuilder*()  
   .setPosterId(resultSet.getInt("POSTER\_ID"))  
   .setNameOfPoster(resultSet.getString("NAME\_OF\_POSTER"))  
   .setPostText(resultSet.getString("POST\_TEXT"))  
   .setPostCategory(request.getPostCategory())  
   .setWhichHouse(resultSet.getInt("WHICH\_HOUSE"))  
   .setHasPhoto(hasPhoto)  
   .build()  
   )  
   .setLiked(isLiked)  
   .build();  
   responseObserver.onNext(post);  
   }  
   responseObserver.onCompleted();  
   } catch (SQLException e) {  
   System.*err*.println(e.getMessage());  
   }  
  }
  + Microserviciul ”getPostsById” - acest microserviciu este folosit pentru regăsirea în baza de date postării distribuite de către un singur utilizator. Aceasta caută în baza de date postării dupa ID-ul utilizatorului în cauză. Este folosită pentru popularea paginilor de profil din aplicație. În interiorul ei, se construiește o interogare SQL care caută dupa ID-ul distribuitorului postări. După, pentru fiecare postare găsită, se produce o nouă interogare, pentru a afla dacă clientul respectiv a apreciat postarea. În final rezultatele interogării se împachetează în tipul de dată Post și se trimite înapoi la client. Mai jos regăsim codul sursă pentru acest microserviciu:

@Override  
public void getPostsById(Posts.PostsRequestByAccountId request, StreamObserver<Posts.Post> responseObserver) {  
 Statement statement = null;  
 try {  
 Connection con = DriverManager.*getConnection*("jdbc:h2:~/Licenta", "root","");  
  
 statement = con  
 .createStatement();  
 String sql = String.*format*("SELECT \* FROM POSTS WHERE POSTER\_ID = '%s' ORDER BY POST\_ID DESC", request.getAccountId());  
 ResultSet resultSet = statement.executeQuery(sql);  
  
 while(resultSet.next()){  
 statement = con.createStatement();  
 ResultSet liked = statement.executeQuery(String.*format*("SELECT liked from Likes WHERE POST\_ID = '%s' AND ACCOUNT\_ID = '%s'",  
 resultSet.getInt("POST\_ID"),resultSet.getInt("POSTER\_ID")));  
 boolean isLiked = false;  
 if(liked.next())  
 isLiked = (liked.getInt("liked") == 1);  
  
 boolean hasPhoto = resultSet.getInt("HAS\_PHOTO") == 1;  
 Posts.Post reply = Posts.Post.*newBuilder*()  
 .setLiked(isLiked)  
 .setPostId(resultSet.getInt("POST\_ID"))  
 .setLitePost(Posts.LitePost.*newBuilder*()  
 .setPosterId(resultSet.getInt("POSTER\_ID"))  
 .setNameOfPoster(resultSet.getString("NAME\_OF\_POSTER"))  
 .setPostText(resultSet.getString("POST\_TEXT"))  
 .setWhichHouse(resultSet.getInt("WHICH\_HOUSE"))  
 .setHasPhoto(hasPhoto)  
 .build())  
 .build();  
  
 responseObserver.onNext(reply);  
 }  
  
 responseObserver.onCompleted();  
 }catch (SQLException e) {  
 throw new RuntimeException(e);  
 }  
}

* + Microserviciul ”likeIt” – aceste microserviciu este folosit pentru aprecierea sau dezaprecierea unei postării existente pe platformă. Se crează o interogare SQL care caută în baza de date dacă postarea este apreciată sau nu. Dupa care starea acestei informați este schimbată:
    - Dacă nu era apreciată, se apreciază
    - Dacă era apreciată, acum nu o să mai fie

@Override  
public void likeIt(Posts.SendLike request, StreamObserver<Posts.PostingReply> responseObserver) {  
  
 Statement statement;  
 try {  
 Connection con = DriverManager.*getConnection*("jdbc:h2:~/Licenta", "root","");  
  
 statement = con  
 .createStatement();  
  
 String verifyExistenceSql = String.*format*("Select \* From Likes Where POST\_ID = '%s' AND ACCOUNT\_ID = '%s'",  
 request.getPostId(),  
 request.getAccountId());  
 ResultSet resultSet = statement.executeQuery(verifyExistenceSql);  
 if(resultSet.next()){  
 String sql = String.*format*("Update Likes Set Liked = '%s' Where POST\_ID = '%s' AND ACCOUNT\_ID = '%s'",  
 parseBooleanToInt(request.getLiked()),  
 request.getPostId(),  
 request.getAccountId());  
 statement = con.createStatement();  
 statement.executeUpdate(sql);  
 }  
 else {  
 String sql = String.*format*("INSERT INTO Likes VALUES('%s', '%s', '%s')",  
 request.getPostId(),  
 request.getAccountId(),  
 parseBooleanToInt(request.getLiked()));  
 statement = con.createStatement();  
 statement.executeUpdate(sql);  
 }  
 responseObserver.onNext(Posts.PostingReply.*newBuilder*().setSuccessful(true).build());  
 responseObserver.onCompleted();  
  
 } catch (SQLException e) {  
 System.*err*.println(e.getMessage());  
 responseObserver.onNext(Posts.PostingReply.*newBuilder*().setSuccessful(false).build());  
 responseObserver.onCompleted();  
 }  
}

* + Microserviciul ”sendComment” – folosit pentru a adăuga un comentariu la o postare deja existentă pe platformă. Atunci cănd utilizatorul intră în secțiunea de comentarii, acesta are opțiunea de a adăuga unul personal. Atunci când acesta o face, acest microserviciu se ocupă de aceasta. Acesta inserează printr-o comandă SQL o nouă intrare în baza de date, în tabelul de comentarii.

@Override  
public void sendComment(Posts.LiteCommentary request, StreamObserver<Posts.PostingReply> responseObserver) {  
  
 Statement statement;  
 try{  
 Connection con = DriverManager.*getConnection*("jdbc:h2:~/Licenta", "root","");  
 statement = con.createStatement();  
 String sql = String.*format*("INSERT INTO COMMENT (COMMENTATOR\_ID, POST\_ID, NAME\_OF\_COMMENTATOR, COMMENT\_TEXT) VALUES ('%s','%s','%s','%s')",  
 request.getCommentatorId(),  
 request.getPostId(),  
 request.getNameOfCommentator(),  
 request.getCommentText()  
 );  
  
 int resultSet = statement.executeUpdate(sql);  
  
 responseObserver.onNext(Posts.PostingReply.*newBuilder*().setSuccessful(true).build());  
 responseObserver.onCompleted();  
  
 } catch (SQLException e) {  
  
 System.*err*.println(e);  
 responseObserver.onNext(Posts.PostingReply.*newBuilder*().setSuccessful(true).build());  
 }  
}

* + Microserviciul ”getComments” – acest microserviciu se ocupă de regăsirea comentariilor unui anumit post în baza de date si de trimiterea acestora înapoi la client. Request-ul conține id-ul postului respectiv, și se crează o interogare SQL bazat pe el. După ce comentariile au fost regăsite, ele sunt propagate înapoi spre client, spre a fi afișate.

Mai jos regasim codul sursă pentru acest microserviciu:

@Override  
public void getComments(Posts.CommentariesRequest request, StreamObserver<Posts.Commentary> responseObserver) {  
  
 Statement statement;  
 try{  
 Connection con = DriverManager.*getConnection*("jdbc:h2:~/Licenta", "root","");  
 statement = con.createStatement();  
 String sql = String.*format*("SELECT \* FROM COMMENT WHERE POST\_ID = '%s' ORDER BY COMMENT\_ID DESC",  
 request.getPostId());  
  
 ResultSet resultSet = statement.executeQuery(sql);  
  
 while(resultSet.next()){  
 Posts.Commentary.Builder replyBuilder = Posts.Commentary.*newBuilder*()  
 .setCommentId(resultSet.getInt("COMMENT\_ID"))  
 .setLiteCommentary(Posts.LiteCommentary.*newBuilder*()  
 .setCommentatorId(resultSet.getInt("COMMENTATOR\_ID"))  
 .setPostId(resultSet.getInt("POST\_ID"))  
 .setNameOfCommentator(resultSet.getString("NAME\_OF\_COMMENTATOR"))  
 .setCommentText(resultSet.getString("COMMENT\_TEXT"))  
 .build());  
 responseObserver.onNext(replyBuilder.build());  
 }  
 responseObserver.onCompleted();  
  
  
 } catch (SQLException e) {  
 System.*err*.println(e);  
 }  
}

* + Microserviciul ”postIt” – se ocupă cu distribuirea unei noi postări, dupa ce clientul a trimis cererea către server. Postările se trimit în două părti către server, iar prima dată se primește informația text despre ea, spre a fi inserată în baza de date. După ce informația a fost inserată cu succes, se interogheaza baza de date pentru a afla id-ul postului nou inserat, pentru a fi trimis ca răspuns înapoi la client.

Mai jos putem găsi codul sursă pentru microserviciul postIt:

@Override  
public void postIt(Posts.LitePost request, StreamObserver<Posts.PostingReply> responseObserver) {  
 //Get the post from the client and store it in the database  
 Statement statement = null;  
 try {  
 Connection con = DriverManager.*getConnection*("jdbc:h2:~/Licenta", "root","");  
  
 statement = con  
 .createStatement();  
  
 String sql = String.*format*("INSERT INTO posts (POSTER\_ID,NAME\_OF\_POSTER,POST\_TEXT,POST\_CATEGORY,WHICH\_HOUSE,WHICH\_CAMPUS,HAS\_PHOTO)"  
 + "VALUES ('%s','%s','%s','%s','%s','%s','%s');",  
 request.getPosterId(),  
 request.getNameOfPoster(),  
 request.getPostText(),  
 parsePostCategoryToString(request.getPostCategory()),  
 request.getWhichHouse(),  
 parseCampusToString(request.getWhichCampus()),  
 parseBooleanToInt(request.getHasPhoto()));  
  
 int resultSet = statement.executeUpdate(sql);  
 sql = String.*format*("SELECT POST\_ID FROM posts WHERE POSTER\_ID = '%s' AND NAME\_OF\_POSTER = '%s' AND " +  
 "POST\_TEXT = '%s' AND POST\_CATEGORY = '%s' AND WHICH\_HOUSE = '%s' AND WHICH\_CAMPUS = '%s'",  
 request.getPosterId(),  
 request.getNameOfPoster(),  
 request.getPostText(),  
 parsePostCategoryToString(request.getPostCategory()),  
 request.getWhichHouse(),  
 parseCampusToString(request.getWhichCampus()));  
 statement = con.createStatement();  
 ResultSet set = statement.executeQuery(sql);  
 set.next();  
 int post\_id = set.getInt("POST\_ID");  
 responseObserver.onNext(Posts.PostingReply.*newBuilder*().setSuccessful(true).setPostId(post\_id).build());  
 responseObserver.onCompleted();  
  
 } catch (SQLException e) {  
 System.*err*.println(e.getMessage());  
 responseObserver.onNext(Posts.PostingReply.*newBuilder*().setSuccessful(false).build());  
 responseObserver.onCompleted();  
 }  
}

3.5.2.3.2 ImagesService

În continuare, vom vorbi despre ImagesService. Singurele scopuri ale acestui serviciu este transferul de imagini, în ambele sensuri ale fluxului client-server. Acesta transferă imaginile prin pulsuri de biți, supranumiți chunk-uri, cu dimensiuni de maxim 1024 de biți, sau un kilobyte. Acest serviciu este folosit în interiorul întregii aplicații, ori pentru a încărca pozele de profil ale utilizatorilor, a primi pozele din interiorul postărilor (lucrând practic împreuna cu serviciul PostsService), sau pentru a trimite serverului poze si imagini. Acest serviciu are doar 2 microservici în interiorul său:

* + Microserviciul ”uploadImage” – acest microserviciu este folosit pentru a transmite imagini de la client către server, pentru a fi folosite mai târziu în diferite moduri, fie ca și poze pentru postării sau ca și poze de profil pentru utilizatori.

@Override  
public StreamObserver<Images.ImageTransfer> uploadImage(StreamObserver<Images.Empty> responseObserver) {  
 return new StreamObserver<Images.ImageTransfer>() {  
 @Override  
 public void onNext(Images.ImageTransfer uploadImageRequest) {  
 if(uploadImageRequest.getDataCase() == Images.ImageTransfer.DataCase.*IMAGE\_INFO*){  
 imageInfo = uploadImageRequest.getImageInfo();  
 imageData = new ByteArrayOutputStream();  
  
 System.*out*.println("uploadImage:onNext");  
 return;  
 }  
 else {  
 ByteString chunkData = uploadImageRequest.getChunkData();  
 if(imageData != null){  
 try {  
 chunkData.writeTo(imageData);  
 } catch (IOException e) {  
 return;  
 }  
 }  
 }  
  
 }  
 @Override  
 public void onError(Throwable throwable) {  
 System.*out*.println("uploadImage:onError + " + throwable.getMessage());  
 }  
 @Override  
 public void onCompleted() {  
  
 //save the image  
 try {  
 saveImage.save(imageInfo,imageData);  
  
 } catch (IOException e) {  
  
 throw new RuntimeException(e);  
 }  
 responseObserver.onNext(Images.Empty.*newBuilder*().build());  
 responseObserver.onCompleted();  
 }  
 };  
}

* + Microserviciul ”downloadImage” – acest microserviciu este folosit pentru a propaga imagini stocate pe server, acolo unde este nevoie de ele, fie în postări, fie ca și poze de profil. Acesta primește un request pentru o anumită poza, și transmite înapoi poza respectivă, dacă ea există stocată pe server.

@Override  
public void downloadImage(Images.ImageInfo request, StreamObserver<Images.ImageTransfer> responseObserver) {  
  
 ServerCallStreamObserver scso =(ServerCallStreamObserver)responseObserver;  
 scso.setOnCancelHandler(() -> {  
 System.*out*.println("The stream was cancelled");  
 });  
  
 String imagePath;  
 if(request.getImageUsage() == Images.ImageUsage.*POST\_PICTURE*)  
 imagePath = String.*format*("'%s'/'%s'\_'%s'\_'%s''%s'",  
 imageFolder,request.getAccountId(),request.getPostId(),request.getImageUsage(),request.getImageType());  
 else  
 imagePath = String.*format*("'%s'/'%s'\_'%s''%s'",  
 imageFolder,request.getAccountId(),request.getImageUsage(),request.getImageType());  
  
 imagePath = imagePath.replaceAll("'","");  
  
 try {  
 FileInputStream fileInputStream = new FileInputStream(imagePath);  
  
 try {  
 responseObserver.onNext(Images.ImageTransfer.*newBuilder*().setImageInfo(request).build());  
 byte[] buffer = new byte[1024]; // 1KB  
 while (true) {  
 int n = fileInputStream.read(buffer);  
 if (n <= 0)  
 break;  
  
 Images.ImageTransfer response = Images.ImageTransfer.*newBuilder*()  
 .setChunkData(ByteString.*copyFrom*(buffer, 0, n))  
 .build();  
 responseObserver.onNext(response);  
 }  
  
 } catch (Exception e) {  
 //throw new RuntimeException(e);  
 System.*err*.println("Something happened in the download image function");  
 }  
 responseObserver.onCompleted();  
 } catch (FileNotFoundException e) {  
 System.*err*.println("Error: " + imagePath + " file not found");  
 }  
}

3.5.2.3.3 LoginService

Serviciul LoginService are un singur scop, acela de a facilita autentificarea pe platformă. Acesta primește un request de la client, care conține datele de logare, adică e-mail-ul instituțional și parola, și verifică dacă există un astfel de cont în baza de date. În cazul negativ, acesta returnează un răspuns care avertizează clientul de eșecul autentificări și motivele pentru care ar putea fi eșuat. În caz contrar, serviciul returnează toate datele asociate contului respectiv, pentru ca aceste date să fie folosite mai târziu de către client, în crearea de alte request-uri.

În serviciul de autentificare avem un singur microserviciu:

* + Microserviciul ”login” – acesta facilitează funcția de autentificare în platformă, așa cum este descris mai sus. În continuare putem observa codul sursă al acestui microserviciu:

@Override  
public void login(Login.LoginRequest request, StreamObserver<Login.LoginReply> responseObserver) {  
 System.*out*.println("Somebody is trying to log in");  
 Login.LoginReply.Builder response = Login.LoginReply.*newBuilder*();  
 Statement statement = null;  
 try {  
 Connection con = DriverManager.*getConnection*("jdbc:h2:~/Licenta", "root","");  
  
 statement = con  
 .createStatement();  
  
 String sql = String.*format*("SELECT id, complex, house FROM accounts WHERE accounts.email = '%s' AND accounts.password = '%s'",  
 request.getEmail(),request.getPassword());  
  
 ResultSet resultSet = statement.executeQuery(sql);  
 if(resultSet.next())  
 {  
 Login.Account account = Login.Account.*newBuilder*()  
 .setId(resultSet.getInt("id"))  
 .setEmail(request.getEmail())  
 .setPassword(request.getPassword())  
 .setCampus(parseStringToCampus(resultSet.getString("complex")))  
 .setHouse(resultSet.getInt("house"))  
 .build();  
  
 response.setSuccessful(true);  
 response.setAccount(account);  
 }  
 else  
 response.setSuccessful(false);  
  
 responseObserver.onNext(response.build());  
 responseObserver.onCompleted();  
  
 } catch (SQLException e) {  
 System.*err*.println(e.getMessage());  
 }  
}

3.5.2.3.4 ChatHistoryService

Un alt serviciu care îndeplinește un singur scop, de a regăsi și trimite către client toate sesiunile de chat recente pe care le-a întreținut. Acesta primește un request de la client, care conține doar ID-ul de cont al clientului, și care preia din baza de date toate intrările din tabelul de ”CHATS” care conțin în una dintre pozițiile acestuia ID-ul clientului care a pornit serviciul. În interiorul său avem prezent doar un microserviciu:

* + Microserviciul ”getChatHistory” – al cărui scop l-am enunțat mai sus.

Mai jos putem observa codul sursă al acestui microserviciu:

@Override  
public void getChatHistory(ChatMenu.ChatHistoryRequest request, StreamObserver<ChatMenu.ChatHistoryReply> responseObserver) {  
 //the request contains the userId  
 //use that to get the whole chat history  
 Statement statement = null;  
 try {  
 Connection con = DriverManager.*getConnection*("jdbc:h2:~/Licenta", "root","");  
  
 statement = con  
 .createStatement();  
  
 String sql = String.*format*("SELECT \* FROM CHATS WHERE FIRST\_ID = '%s' OR SECOND\_ID = '%s'",  
 request.getUserId(),request.getUserId());  
  
 ResultSet resultSet = statement.executeQuery(sql);  
 if(resultSet.next())  
 {  
 //the first ID is the client, so we need to query on the second  
 statement = con.createStatement();  
 if(resultSet.getInt("FIRST\_ID") == request.getUserId()){  
 sql = String.*format*("SELECT EMAIL FROM ACCOUNTS WHERE ID = '%s'",  
 resultSet.getInt("SECOND\_ID"));  
 }  
 //The second ID is the client, so we need to query on the first  
 else {  
 sql = String.*format*("SELECT EMAIL FROM ACCOUNTS WHERE ID = '%s'",  
 resultSet.getInt("FIRST\_ID"));  
 }  
 ResultSet nameResult = statement.executeQuery(sql);  
 nameResult.next();  
 int otherId;  
 if(resultSet.getInt("FIRST\_ID") == request.getUserId())  
 otherId = resultSet.getInt(("SECOND\_ID"));  
 else  
 otherId = resultSet.getInt(("FIRST\_ID"));  
 String email = nameResult.getString("EMAIL");  
 String name = email.substring(0,email.indexOf(".")) + " " + email.substring(email.indexOf(".")+1,email.indexOf("@"));  
 ChatMenu.ChatHistoryReply reply = ChatMenu.ChatHistoryReply.*newBuilder*()  
 .setChatId(resultSet.getInt("CHAT\_ID"))  
 .setName(name)  
 .setAccountId(otherId)  
 .build();  
 responseObserver.onNext(reply);  
 }  
 responseObserver.onCompleted();  
  
 } catch (SQLException e) {  
 System.*err*.println(e.getMessage());  
 }  
}

3.5.2.3.5 SearchAndChatService

În continuare, SearchAndChatService este serviciul care facilitează funcția de căutare a altor utilizatori prezenți pe platformă, pentru a interacționa cu aceștia. Acesta poate căuta persoane în baza de date, după numele acestora, și returneaza rezultatele înapoi la client pentru afișare și utilizare, ori pentru a începe noi sesiuni de mesagerie instant, sau pentru a vizualiza pagina de profil a utilizatorului selectat în urma căutări.

În acest serviciu regăsim patru microservici, pe care le vom prezenta mai jos:

* + Microserviciul ”getAccountInfo” – este un microserviciu prin care clientul cere informații despre contul unui alt utilizator, decât cel autentificat, pentru uz în crearea altor request-uri sau pentru afișarea datelor pe interfața grafică a clientului, cum ar fi numele.

Codul sursă al acestui microserviciu este:

@Override  
public void getAccountInfo(SearchAndChat.AccountRequest request, StreamObserver<Login.Account> responseObserver) {  
 Statement statement = null;  
 try {  
 Connection con = DriverManager.*getConnection*("jdbc:h2:~/Licenta", "root", "");  
  
 statement = con  
 .createStatement();  
  
 String[] name = request.getName().split(" ");  
 String sql = "SELECT \* FROM ACCOUNTS WHERE EMAIL LIKE ''%s'.'%s'@%";  
 String query = String.*format*(sql,name[0],name[1]);  
  
  
 ResultSet resultSet = statement.executeQuery(sql);  
 if(resultSet.next()){  
 Login.Account account = Login.Account.*newBuilder*()  
 .setId(resultSet.getInt("id"))  
 .setEmail(resultSet.getString("EMAIL"))  
 .setCampus(parseStringToCampus(resultSet.getString("complex")))  
 .setHouse(resultSet.getInt("house"))  
 .build();  
  
 responseObserver.onNext(account);  
 }  
  
 responseObserver.onCompleted();  
  
 } catch (SQLException e) {  
 System.*err*.println(e.getMessage());  
 }  
}

* + Microserviciul ”getAccountIdFromChat” – este un microserviciu prin care obținem ID-ul unei alte persoane din cadrul tabelului de sesiuni de chat-uri, pentru eventuală începere a unei noi sesiuni de conversație cu acesta. Codul sursă este:
* @Override  
  public void getAccountIdFromChat(SearchAndChat.RequestId request, StreamObserver<SearchAndChat.ReplyId> responseObserver) {  
   Statement statement = null;  
   try {  
   Connection con = DriverManager.*getConnection*("jdbc:h2:~/Licenta", "root", "");  
    
   statement = con  
   .createStatement();  
    
    
   String sql = String.*format*("SELECT \* FROM CHATS WHERE CHAT\_ID = '%s'",  
   request.getChatId());  
    
   ResultSet resultSet = statement.executeQuery(sql);  
   if(resultSet.next()){  
   int id;  
   if(resultSet.getInt("FIRST\_ID") == request.getOwnAccountId())  
   id = resultSet.getInt("SECOND\_ID");  
   else  
   id = resultSet.getInt("FIRST\_ID");  
   SearchAndChat.ReplyId replyId = SearchAndChat.ReplyId.*newBuilder*()  
   .setAccountId(id)  
   .build();  
   responseObserver.onNext(replyId);  
   }  
    
   responseObserver.onCompleted();  
    
   } catch (SQLException e) {  
   System.*err*.println(e.getMessage());  
   }  
  }
  + Microserviciul ”searchForPeople” – este microserviciul care facilitează căutarea de utilizatori prezenți pe platforma noastră, prin intermediul interogării bazei de date, extragându-se numele utilizatorilor din adresa acestor de e-mail instituțională. Rezultatele sunt după trimise înapoi la client pentru procesare și afișare în timp real. Codul sursă al acestui microserviciu este:
* @Override  
  public void searchForPeople(SearchAndChat.SearchRequest request, StreamObserver<SearchAndChat.SearchReply> responseObserver) {  
   Statement statement = null;  
   try {  
   Connection con = DriverManager.*getConnection*("jdbc:h2:~/Licenta", "root", "");  
    
   statement = con  
   .createStatement();  
    
    
   String sql = "SELECT \* FROM ACCOUNTS WHERE EMAIL LIKE '%" + request.getText() + "%'";  
    
    
   ResultSet resultSet = statement.executeQuery(sql);  
    
   while (resultSet.next()) {  
   String email = resultSet.getString("EMAIL");  
   String name = email.substring(0,email.indexOf(".")) + " " + email.substring(email.indexOf(".")+1,email.indexOf("@"));  
    
   SearchAndChat.SearchReply reply = SearchAndChat.SearchReply.*newBuilder*()  
   .setName(name)  
   .setOtherAccountId(resultSet.getInt("ID"))  
   .build();  
   responseObserver.onNext(reply);  
   System.*out*.println(String.*format*("Sent: '%s'",reply.getName()));  
   }  
   responseObserver.onCompleted();  
    
   } catch (SQLException e) {  
   System.*err*.println(e.getMessage());  
   }  
  }
  + Microserviciul ”startNewChat” – acest microserviciu facilitează începerea unei sesiuni de mesagerie instant cu o persoană cu care nu am mai avut contact în această facilitate a platformei, astfel inserându-se în baza de date toate datele necesare susțineri acestei funcționalități. Mai jos putem găsi codul sursă al acestui microserviciu:

@Override  
public void startNewChat(SearchAndChat.StartChatRequest request, StreamObserver<SearchAndChat.StartChatReply> responseObserver) {  
 Statement statement = null;  
 try {  
 Connection con = DriverManager.*getConnection*("jdbc:h2:~/Licenta", "root", "");  
  
 statement = con  
 .createStatement();  
  
  
 String sql = String.*format*("SELECT \* FROM CHATS WHERE (FIRST\_ID = '%s' AND SECOND\_ID = '%s') OR (FIRST\_ID = '%s' AND SECOND\_ID = '%s')",  
 request.getOwnId(),request.getOtherAccountId(),request.getOtherAccountId(),request.getOwnId());  
  
  
 ResultSet resultSet = statement.executeQuery(sql);  
  
 if(resultSet.next()){  
 SearchAndChat.StartChatReply reply = SearchAndChat.StartChatReply.*newBuilder*()  
 .setChatId(resultSet.getInt("CHAT\_ID"))  
 .build();  
 responseObserver.onNext(reply);  
 responseObserver.onCompleted();  
 }  
 else {  
 sql = String.*format*("INSERT INTO CHATS (FIRST\_ID,SECOND\_ID) VALUES ('%s','%s')",  
 request.getOwnId(),request.getOtherAccountId());  
 statement = con.createStatement();  
 statement.executeUpdate(sql);  
  
 sql = String.*format*("SELECT \* FROM CHATS WHERE FIRST\_ID = '%s' AND SECOND\_ID = '%s'",  
 request.getOwnId(),request.getOtherAccountId());  
 statement = con.createStatement();  
 resultSet = statement.executeQuery(sql);  
 resultSet.next();  
 responseObserver.onNext(SearchAndChat.StartChatReply.*newBuilder*().setChatId(resultSet.getInt("CHAT\_ID")).build());  
 responseObserver.onCompleted();  
  
 }  
  
 } catch (SQLException e) {  
 System.*err*.println(e.getMessage());  
 }  
}

3.5.2.3.6 MessagingService

Acest serviciu este cel care facilitează funcționalitatea de mesagerie instant a platformei, prin crearea de canele directe între utilizatori aplicației, pentru a primi mesajele în timp real. În esență este un serviciu simplu si eficient, care pe lângă transmiterea mesajelor, stochează mesajele în interiorul bazei de date, pentru a crea un istoric de mesaje în interiorul sesiunilor de mesagerie.

Acest serviciu se prezintă cu două microservici pe care le vom prezenta mai jos:

* + - Microserviciul ”getMessagingHistory” – este microserviciul care la momentul reinițiere unei sesiune de mesagerie, acesta preia toate mesajele din istoricul conversației și le trimite înapoi la clienți pentru afișare, astfel se crează un istoric de mesaje, experiența de mesagerie fiind resimțită continuă și fluidă.

Codul sursă al acestui microserviciu este regăsit mai jos:

@Override  
public void getMessagingHistory(Messaging.MessageHistoryRequest request, StreamObserver<Messaging.Message> responseObserver) {  
 //The request contains only the chatId  
 //Just send the whole chat history for this chat with the help of the $ChatId  
 Statement statement = null;  
 try {  
 Connection con = DriverManager.*getConnection*("jdbc:h2:~/Licenta", "root", "");  
  
 statement = con  
 .createStatement();  
  
  
 String sql;  
 if (request.getLastMessageId() == 0)  
 sql = String.*format*(*normalSql*,  
 request.getChatId());  
 else sql = String.*format*(*normalSql* + " AND MESSAGE\_ID > '%s'",  
 request.getChatId(), request.getLastMessageId());  
  
  
 ResultSet resultSet = statement.executeQuery(sql);  
  
 while (resultSet.next()) {  
 String s = resultSet.getString("MESSAGE\_TEXT");  
 Messaging.Message reply = Messaging.Message.*newBuilder*()  
 .setMessageId(resultSet.getInt("MESSAGE\_ID"))  
 .setChatId(resultSet.getInt("CHAT\_ID"))  
 .setMessageOwnerId(resultSet.getInt("OWNER\_ID"))  
 .setMessage(resultSet.getString("MESSAGE\_TEXT"))  
 .build();  
  
 responseObserver.onNext(reply);  
 }  
 responseObserver.onCompleted();  
  
 } catch (SQLException e) {  
 System.*err*.println(e.getMessage());  
 }  
}

* + - Microserviciul ”sendReceiveMessage” – este microserviciul care facilitează funcționalitate de mesagerie instantă a platformei, aceasta creând canale directe de transmitere a datelor între utilizatori aplicației. Aceasta memorează toate canalele de mesagerie instant într-un HashMap, în care cheia de acces este ID-ul chatului, iar valoarea este o instanță a clasei ”ChatSesion”. Codul sursă al acestui microserviciu este:

@Override  
public StreamObserver<Messaging.Message> sendReceiveMessage(StreamObserver<Messaging.MessageFromServer> responseObserver) {  
 // return super.sendReceiveMessage(responseObserver);  
 //While you are on the inside of the chat the bilateral stream will be a continue one  
  
 final int[] chat\_id = new int[1];  
 final int[] owner\_id = new int[1];  
  
 return new StreamObserver<Messaging.Message>() {  
 @Override  
 public void onNext(Messaging.Message message) {  
 //receiving data from client  
  
 if(message.getMessage().equals("")){  
 if(*observers*.containsKey(message.getChatId())) {  
 *observers*.get(message.getChatId()).setOnline(message.getMessageOwnerId(),responseObserver);  
 }  
 else{  
 ChatSession newSession = new ChatSession();  
 newSession.setOnline(message.getMessageOwnerId(),responseObserver);  
 *observers*.put(message.getChatId(),newSession);  
 }  
 }  
 else{  
 Statement statement = null;  
 chat\_id[0] = message.getChatId();  
 owner\_id[0] = message.getMessageOwnerId();  
 try {  
 Connection con = DriverManager.*getConnection*("jdbc:h2:~/Licenta", "root", "");  
  
 statement = con  
 .createStatement();  
  
 String sql = String.*format*("INSERT INTO MESSAGES (CHAT\_ID, OWNER\_ID, MESSAGE\_TEXT) VALUES ('%s','%s','%s')",  
 message.getChatId(), message.getMessageOwnerId(),message.getMessage());  
  
 int resultSet = statement.executeUpdate(sql);  
  
 } catch (SQLException e) {  
 throw new RuntimeException(e);  
 }  
   
 *observers*.get(message.getChatId()).sendMessage(Messaging.MessageFromServer.*newBuilder*().setMessage(message).build());  
 System.*out*.println(String.*format*("Got a message from: '%s' : '%s'", message.getMessageOwnerId(), message.getMessage()));  
 }  
  
 }  
 @Override  
 public void onError(Throwable throwable) {  
  
 if (*observers*.containsKey(chat\_id[0])) {  
 *observers*.get(chat\_id[0]).setOffline(owner\_id[0]);  
 }  
 System.*err*.println(throwable.getMessage());  
 }  
 @Override  
 public void onCompleted() {  
 responseObserver.onCompleted();  
 if(*observers*.get(chat\_id[0]) != null) {  
 *observers*.get(chat\_id[0]).setOffline(owner\_id[0]);  
 }  
 }  
  
 };  
}

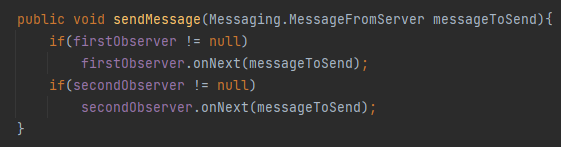
3.5.2.4 Clasa ChatSession

Clasa ChatSession memorează datele despre o sesiune de chat activă. Aceasta este folosita de serviciul de mesagerie instant. Câmpurile acesteia sunt:

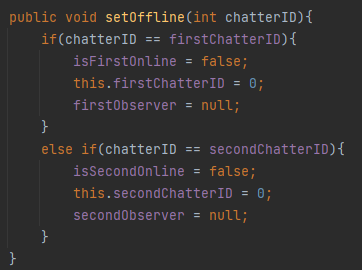
* firstChatterID – memorează ID-ul primului cont de utilizator al sesiuni de mesagerie
* secondChatterID – memorează ID-ul celui de al doilea cont de utilizator al sesiuni de mesagerie
* firstObserver – memorează canalul de transmitere a mesajelor către primul utilizator
* secondObserver – memorează canalul de transmitere a mesajelor către cel de al doilea utilizator
* isFirstOnline – memorează starea primului utilizator
* isSecondOnline – memorează starea celui de al doilea utilizator

Această clasa conține funcți ajutătoare, cum ar fi:

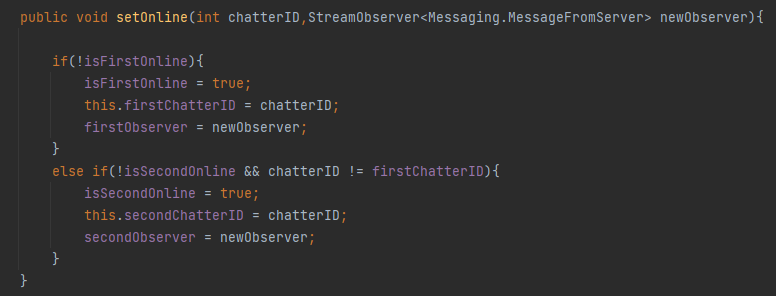
* „sendMessage” – aceasta funcție propagă un mesaj către utilizatori care sunt online din această sesiune de mesagerie



* „setOffline” – închide conexiunea către unul dintre utilizatori mesageriei, atunci când acesta închide sesiunea



* „setOnline” – memoreaza conexiunea către un utilizator pentru propagarea mesajelor atunci când acesta pornește sesiunea de mesagerie



3.6 Baza de date pentru aplicația de tip server

Pentru crearea bazei de date am folosit interfața web oferită de H2 Database Engine. Aici m-am utilizat de instrucțiuni din limbajul SQL pentru crearea și corelarea tabelelor de care am avut nevoie. În interiorul bazei de date dispunem de șase tabele:

* ACCOUNTS – pentru memorarea informațiilor conturilor instituționale
* CHATS – pentru memorarea sesiunilor de mesagerie instant existente
* COMMENT – pentru stocarea comentariilor postate în cadrul postărilor
* LIKES – unde memorăm aprecierile date de utilizatori postărilor
* MESSAGES – aici stocăm mesajele trimise prin intermediul mesageriei instant
* POSTS – unde memorăm postările distribuite prin intermediul platformei

În interiorul tabelei ACCOUNTS avem următoarele câmpuri:

* ID – memorează ID-ul utilizatorului
* EMAIL – memorează e-mail-ul utilizatorului
* PASSWORD – memorează parola utilizatorului
* COMPLEX – memorează complexul în care utilizatorul este cazat
* HOUSE – memorează numărul căminului în care utilizatorul este cazat

În tabelul CHATS avem următoarele câmpuri:

* CHAT\_ID – memorează ID-ul sesiuni de mesagerie
* FIRST\_ID – memorează ID-ul primului participant la sesiunea de mesagerie
* SECOND\_ID – memorează ID-ul celui de al doilea participant la sesiunea de mesagerie

În tabelul COMMENT avem următoarele câmpuri:

* COMMENT\_ID, memorează ID-ul comentariului
* COMMENTATOR\_ID, memorează ID-ul utilizatorului care a postat comentariul
* POST\_ID, memorează ID-ul postului unde comentariul a fost postat
* NAME\_OF\_COMMENTATOR, memorează numele celui care a comentat
* COMMENT\_TEXT, memorează textul comentariului postat

În tabelul LIKES avem următoarele câmpuri:

* POST\_ID, memorează ID-ul postului apreciat
* ACCOUNT\_ID, memorează ID-ul utilizatorului care a apreciat
* LIKED, memorează dacă postul este apreciat sau nu

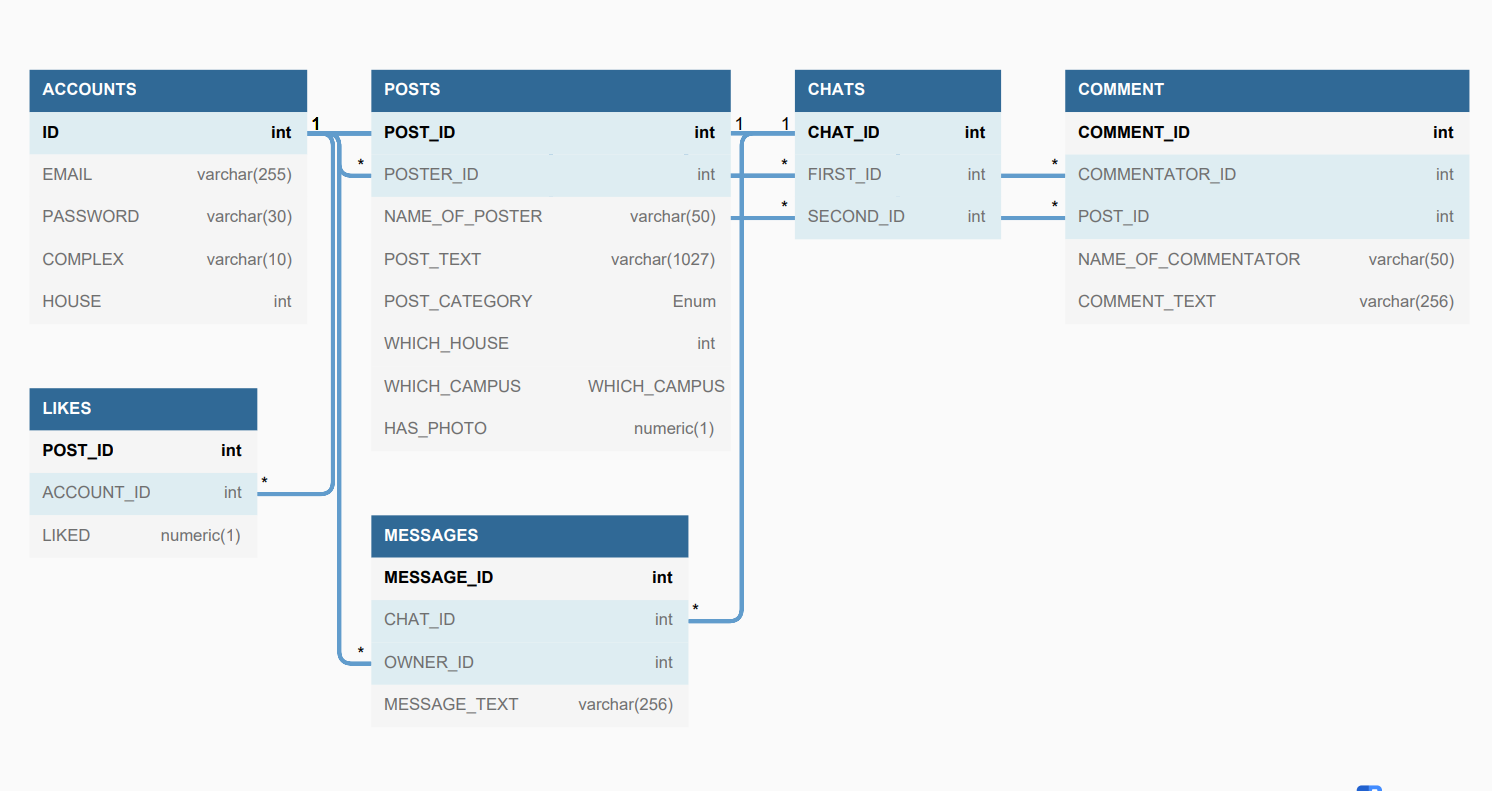
În tabelul MESSAGES avem următoarele câmpuri:

* MESSAGE\_ID, memorează ID-ul mesajului
* CHAT\_ID, memorează ID-ul sesiuni de mesagerie în care mesajul a fost trimis
* OWNER\_ID, memorează ID-ul autorului de mesaj
* MESSAGE\_TEXT, memorează textul mesajului trimis

În tabelul POSTS avem următoarele câmpuri:

* POST\_ID, memorează ID-ul postări
* POSTER\_ID, memorează ID-ul utilizatorului care a distribuit postarea
* NAME\_OF\_POSTER, memorează numele utilizatorului care a distribuit postarea
* POST\_TEXT, memorează textul postări, dacă acesta există
* POST\_CATEGORY, memorează categoria în care această postare va fi disponibilă (ALL, CAMPUS, HOUSE)
* WHICH\_HOUSE, memorează pentru ce cămin va fi aceasta postare disponibilă
* WHICH\_CAMPUS, memorează pentru ce campus va fi aceasta postare disponibilă
* HAS\_PHOTO, memorează dacă postarea are sau nu poză, pentru a fi eventual încărcată

Mai jos găsim diagrama relațională a entităților bazei de date:



****4. Dezvoltarea si scalarea proiectului****

Atât aplicația client Android, cât și serverul Java pot fi dezvoltate în continuare. Fie pentru împrospătarea interfeței grafice si adăugarea de noi elemente în interiorul acesteia, fie prin mărirea capacități serverului de transmitere a datelor în moduri mai eficiente. a

4.1 Dezvoltarea aplicației client Android

Unul dintre primele lucruri care ar putea fi dezvoltate pe partea de client este implementarea posibilități de transfer a imaginilor și în partea de mesagerie instant a aplicației. De asemenea aici se mai pot adăuga elemente de mesagerie precum emoticoane, GIF-uri, răspunsuri la mesaje anterioare. De asemenea performanța totala a aplicației poate fi îmbunătațită prin folosirea de noi metode de programare și consumare a datelor pentru a reduce timpul de procesare și afișare a acestora.

4.2 Dezvoltarea aplicației server Java

Aplicația server poate fi îmbunătațită prin crearea de noi microservici specializate pe tipuri de informație din ce în ce mai mici, și combinarea apelurilor acestora în partea de client a platformei. Astfel se va obține o modularizare și reutilizare mai mare a microserviciilor, care ar crește performanța totală.

Concluzie

Proiectul descrie un mediu de studiu al modului în care se dezvoltă o platformă socială, utilizând dezvoltarea unei aplicație de Android care cu ajutorul framework-ului de gRPC comunică cu un server centralizat pentru împărtașirea de date. De asemenea se studiează modul în care sunt dezvoltate aplicațiile în Java, și cum putem stoca date și informații cu ajutorul bazelor de date.

**Platforma create reprezintă doar un prototip pentru o idee mai mare, dar care poate deveni reală și funcțională, o unealtă care se poate dovedi folositoare în ecosistemul studențesc.**

****A picture containing text

Description automatically generated****

****Bibliografie:****

<https://stackoverflow.com/questions/4989182/converting-java-bitmap-to-byte-array>

<https://ro.wikipedia.org/wiki/Re%C8%9Bea_de_socializare>

<https://ro.warbletoncouncil.org/tipos-de-redes-sociales-3308#menu-10>

<https://idsi.md/files/file/prezentari_practica_studenti/Retele%20sociale.pdf>

<https://www.inspectmun.ro/avantajele-si-dezavantajele-retelelor-sociale/>

<https://ro.economy-pedia.com/11030785-advantages-and-disadvantages-of-social-networks>

<https://www.clinicaoananicolau.ro/dependenta-de-internet/>

<https://en.wikipedia.org/wiki/Android_(operating_system)>

<https://en.wikipedia.org/wiki/Android_Studio>

<https://en.wikipedia.org/wiki/IntelliJ_IDEA>

<https://developer.android.com/guide/topics/resources/providing-resources>

<https://en.wikipedia.org/wiki/XML>

<https://developer.android.com/guide/topics/ui/declaring-layout>

<https://ro.wikipedia.org/wiki/SQL>

<https://en.wikipedia.org/wiki/H2_(DBMS)>

<https://www.h2database.com/html/main.html>

<https://en.wikipedia.org/wiki/GRPC>

<https://en.wikipedia.org/wiki/Protocol_Buffers>

<https://stackoverflow.com/questions/22105775/imageview-in-circular-through-xml>

<https://stackoverflow.com/questions/4989182/converting-java-bitmap-to-byte-array>

<https://stackoverflow.com/questions/37523308/when-onbindviewholder-is-called-and-how-it-works>

<https://stackoverflow.com/questions/34969446/grpc-image-upload>

<https://stackoverflow.com/questions/27083091/recyclerview-inside-scrollview-is-not-working>

<https://stackoverflow.com/questions/32159724/scroll-to-top-in-recyclerview-with-linearlayoutmanager>

<https://www.youtube.com/watch?v=DU-q5kOf2Rc>

<https://www.youtube.com/watch?v=rEmFMPv3TsA&t=925s>