

**Universitatea „Politehnica” Timișoara**

**Facultatea de Automatică și Calculatoare**

**Programul de Licență: #tbd**

**See Now**

Apicație de stocare imagini și recunoaștere de persoane

**Lucrare de licență**

Conducător: Student:

Prof. dr. ing Răzvan-Dorel CIOARGĂ Răzvan-Alexandru BORDUZ

Timișoara,

Iunie 2018

Cuprins

[1 Introducere 4](#_Toc482813935)

[1.1 Scopul lucrarii 4](#_Toc482813936)

[1.2 Contextul și domeniul temei 4](#_Toc482813937)

[1.2.1 Tehnologia Android 4](#_Toc482813938)

[1.2.2 Detecție facială 6](#_Toc482813939)

1.3 Tema propriu-zisă 11

[2 Medii și tehnologii de programare folosite 12](#_Toc482813943)

[2.1 Android Studio 10](#_Toc482813944)

[2.2 MySQL Server 11](#_Toc482813945)

[2.3 OpenCV 3.0 13](#_Toc482813946)

[2.4 Ubuntu 16 14](#_Toc482813947)

[2.5 Microsoft Visual Studio 14](#_Toc482813948)

[2.6 Apache2 14](#_Toc482813948)

[2.7 Github 14](#_Toc482813948)

[2.8 Descrierea limbajelor de programare 17](#_Toc482813950)

[2.8.1 Java 15](#_Toc482813951)

[2.8.2 Python & OpenCV 16](#_Toc482813952)

[2.8.3 PHP 17](#_Toc482813953)

[2.8.4 SQL 1](#_Toc482813954)8

[3 Documentație tehnică a aplicației 20](#_Toc482813955)

[3.1 Design 22](#_Toc482813956)

[3.2 Implementare 25](#_Toc482813957)

[3.2.1 Baze de date 25](#_Toc482813951)

[3.2.2 Server 35](#_Toc482813952)

[3.2.3 Algoritmi detecție și recunoaștere facială](#_Toc482813953) 50

[3.2.4 Interfață utilizator 5](#_Toc482813954)6

[3.3 Testare 25](#_Toc482813957)

[4 Concluzii 6](#_Toc482813962)5

[5 Referințe bibliografice 67](#_Toc482813963)

# Introducere

## Scopul lucrării

Aplicația See Now are ca scop aducerea unei unelte în favoarea persoanelor care suferă de anxietate social sau chiar fobie socială. Totodată are ca scop eficientizarea stocării de date ale utilizatorului, oferind o soluție validă de stocare imagini.

Anxietatea socială a fost definită inițial drept o experiență de stres, disconfort, teamă în situații sociale și teama de a primi evaluări negative de la alții. [1]

Impactul socio-economic al tulburării de anxietate social are loc atât asupra celor care suferă, cât și asupra comunității este considerabil. Pentru o persoană cu tulburare de anxietate socială, calitatea vieții este mult redusă: munca, relațiile sociale și persoanele sunt toate afectate. Totodată persoanele care suferă de tulburării de anxietate socială ajung să fie foarte izolate, având dificultăți mari în a iniția o conversație cu o altă persoană necunoscută.

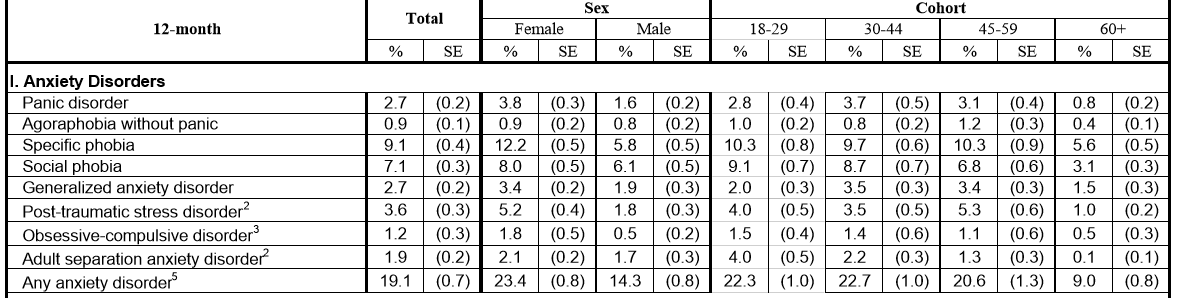
Un studiu al celor de la NIH (*National Institute of Mental Health*) atestă faptul că peste 9% din tinerii între 18-29 de ani suferă de anxietate socială. 

Figure 1. Statistici referitoare tulburări de anxietate[2]

## Contextul și domeniul temei

### Tehnologia Android

Android este o platformă open source concepută pentru dispositive mobile. Este susținută de Google și deținută de Open Handset Alliance.

Android este o plaformă **cuprinzătoare**, conținând tot software-ul necesar pentru un dispozitiv mobil. Întregul conținut, de la module low-level Linux până la biblioteci native este la disponibil.

Android a fost proiectat să ruleze pe tot felul de dispositive fizice, fără a fii dependent de dimensiune, rezoluție, chipset, etc, fiind proiectat să fie porbail, acest lucru fiind realizat datorită nivelului de abstractizare oferit de către nucleul Linux permițând Androidului să fie portat pe o mare varietate de platforme. [4]

Android este un produs software în permanență dezvoltare și îmbunătățire, așadar există o varietate de versiuni ale acestuia ( vezi fig 2 ).

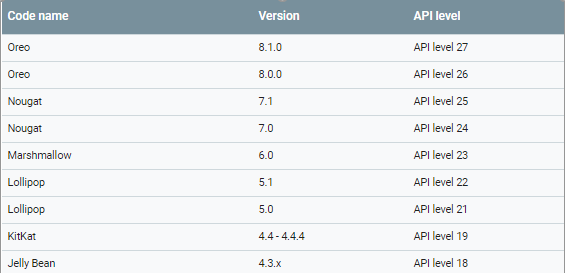


Figure 1.2 Versiuni android și API Level [3]

**Nucleul linux** este folosit de către Android pentru gestionarea memoriei, proceselor, rețele și alte servicii ale sistemului de operare. Aplicațiile nu au acces direct la cadrul linux și acest nivel al nucleului vine împreună cu **nivelul de abstractizare hardware** (HAL)

**Bibliotecile native** reprezintă nivelul următor după nucleul linux. Acestea fiind scrise în C sau C++ și compilate pentru arhitectura specifică telefonului. Cele mai importante librării native fiind: Codecuri media, baze de date SQL, browser engine, manager de suprafață și grafici 2d și 3d.

Tot deasupra nucleului se află și nivelul **Android Runtime** care conține mașina virtuală **Dalvik** și **nucleul librăriilor Java**.Codul scris este compilat în instrucțiuni independente de mașină numite bytecodes, care apoi sunt executate de mașina virtuală Dalvik pe dispozitivul mobil.

Deasupra bibliotecilor native veți găsi **nivelul cadrului aplicație**. Acest nivel oferă uneltele de nivel înalt care sunt folosite pentru a crea aplicații. Cele mai importante părți ale cadrului fiind:

* Managerul de activități, asigură controlul asupra ciclului de viață al aplicațiilor ( vezi figura 4 )
* Furnizori de conținut, încapsulează informația care trebuie distribuită între aplicații
* Managerul de resurse
* Managerul de locație
* Managerul de notificații

Cel mai înalt nivel al arhitecturii Android este **nivelul Aplicațiilor și widgeturilor**. Aplicații precum: Calendar, Contacte, Browser Web și aplicația prezentată în această lucrare: **See Now**.

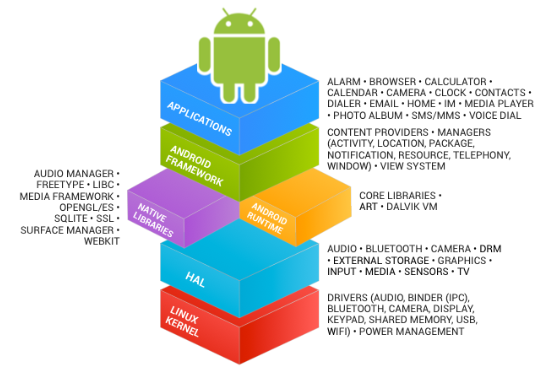
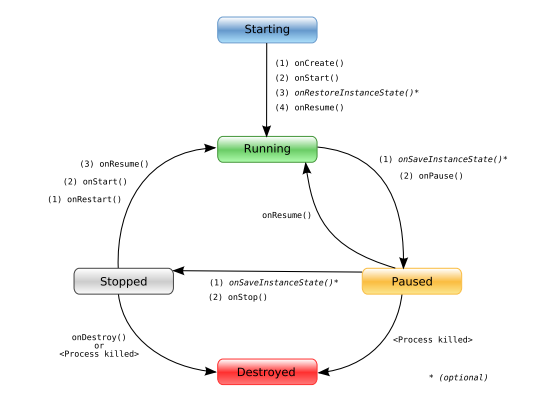


Figure 1.3 Stiva Android [3]

Figure 1.4 Ciclul de viață al unei activități [4]

Combinând toate aceste nivele obținem diagram stivei Android. (vezi fig 1.3).

### Detecție facială

Detecția feței umane are un impact major în aplicații precum supravegherea video,

interfața computerului uman, recunoașterea feței și gestionarea bazelor de date cu imagini. În momentul de față recunoașterea facial este folosită de către Facebook, care îți sugerează zona în care ar putea fi fața unei persoane, pe aplicația de Messenger găsim tot o implementare a algoritmilor de recunoaștere facială pentru efectele și măștile care se aplică automat asupra imaginii.

Sistemele de recunoaștere facială sunt aplicații care pe baza informațiilor obținute în urma aplicării anumitor algoritmi specifici pot clasifica o imagine, atribuindu-i anumite trăsături: sex, vârstă, nume.

**Recunoașterea facială folosind clasificatori de tip cascadă bazați pe caracteristici LBP**

Folosirea metodie LPB implică descompunerea imaginii în mai multe secțiuni, iar pentru fiecare secțiune facem o parcurgere a tuturor pixelilor din secțiunea respective pentru a obține valoarea pixelilor. Pentru fiecare pixel central se verifică valorile pixelilor vecini, dacă pixelul vecin are valoarea mai mare acesta ia valoarea 1, altfel ia valoarea 0. Parcurgerea pixelilor se vace într-un sens stabilit, fie în sensul acelor de ceasornic , fie în sens trigonometric, rezultând un șir de 8 biți. Pentru fiecare secțiune se generează histograma respectivă secțiunii, iar apoi se trece la următoarea. După ce au fost parcurse toate secțiunile, se concatenează histogramele rezultate și obținem o histogramă care conține informațiia utilă unei viitoare clasificări (vezi figura 1.6 ).

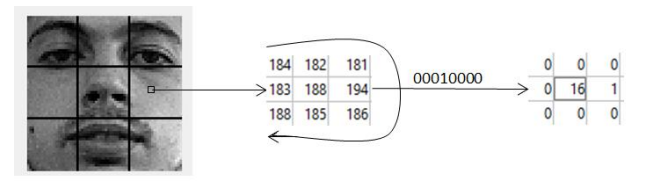


Figure 1.5 Exemplu operator LBP [5]

Un model binar local se numește uniform dacă conține cel mult două tranziții de biți de la 0 la 1 sau invers atunci când șirul binar este considerat circular. Exemplu: 00000000, 00011110 și 10000011 sunt modele uniforme.

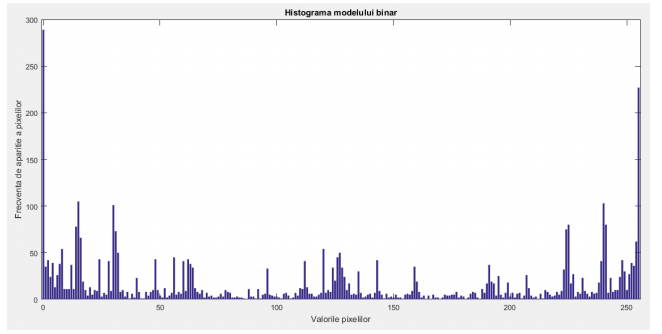


Figure 1.6 Exemplu de histogramă

Pentru creșterea acurateței acestei metode se pot folosi mai multe imagini în procesul de antrenare, totodată putem folosi și imagini care nu se aseamănă cu o față.

Avantajele metodei este faptul că necesită un nivel de calcul scăzut și este o metodă rapidă în comparație cu metoda Fisherfaces, însă marele dezavantaj al acestia fiind totuși acuratețea.

**Analiza discriminatorie lineară** ( engl.Linear Discriminant Analysis - LDA ) este o metodă statică de selecție a caracteristicilor. Se urmărește obținerea unei proiecți în sensul maximizării matricii de covariație între clase, respectiv minimizarea matricii de covariație din interiorul claselor. Această metodă este cel mai des folosită în statistici, recunoașterea de obiecte, precum și în machine learning.

Analiza discriminatorie lineară este similar cu analiza principal a componentelor (PCA) și analiza factorului prin faptul că toate acestea caută combinații liniare de variabile care explică cel mai bine datele. LDA încearcă să modeleze diferența dintre clasele de date, spre deosebire de PCA care nu ține cont de acest aspect.

LDA funcționează atunci când măsurătorile efectuate asupra variabilelor independente pentru fiecare observație sunt cantități continue. În cazul în care se ocupă cu variabile independente categorice, tehnica echivalentă este analiza corespondenței discriminantă. [8]

**Algoritmul LDA**

Fie C persoane fotografiate( clase) cu câte Ni ( i = 1:C ) vectori n-dimensionali în fiecare clasă ( în acest caz dimensiunea n – reprezentând numărul de pixeli )

1. Se calculează matricea de covariație între clase, respectiv matricea de covariație din interiorul clasei.

* **SB** este matricea de covariație între clase :



* **P(ωi)** reprezintă probabilități apriori
* **µ** este media celor C clase, µi este media fiecarei clase
* **SW** este matricea de covariație din interiorul clasei:



1. Se determină valorile proprii ale matricei Sw-1SB  și apoi se ordonează descrescător valorile proprii obținute.



1. Se determină vectorii proprii corespunzătoiri valorilor proprii (Φ𝑖 ):



1. Fiecare vector de intrare X se va proiecta în spațiul LDA după formula: Y = L ⋅ X, unde L = [Φ1, Φ2, Φ3, …, Φn]T și se numește matricea de transformare LDA formată din vectorii proprii determinați anterior ( vezi pasul 3. )

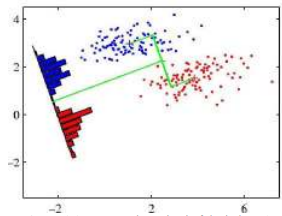


Figure 1.7 Exemplu dreaptă proiecție folosind LDA

**Recunoașterea facială folosind algoritmul Fisherfaces**

Metoda de proiecție are la bază algoritmul LDA al lui Fisher prezentat anterior. Pentru calculul utilizând algoritmul Fisherfaces presupunem că datele sunt distribuite normal, distribuția fiind notată cu 𝑁𝑖(𝜇𝑖 , ∑𝑖), unde μi reprezintă media ,Σi  reprezintă matricea de covarianță și funcția densității de probabilitate f(x|μi,Σi) .

În cazul algoritmului Fisherfaces:

* Matricea de covariație între clase este calculate după formula:



* + Unde μi este media imaginilor din clasa Xi
* Matricea de covariație din interiorul clasei:



* + Unde nj este numărul de imagini dintr-o clasă Ci

Algoritmul Fisherfaces urmărește obținerea unei matrici W ( matrice cu coloane ortonormate ) astfel încât raportul SB și SW să fie maximizat.

Proiecția optimală este calculate după formula:

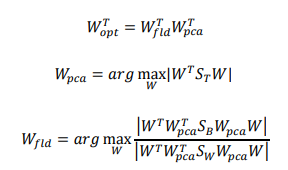


Unde {wi| i=1,2,…m} este un set de vectori proprii ai matricelor SW și SB care corespund celor mai importante m valori proprii {λi| i=1,2,…m}



O problem cu recunoașterea facial este aceea că matricea de dispersie din interiorul clasei are determinantul 0, lucru datorat faptului că matricea de covariație din interiorul clasei are rangul maxim N –n, de aceea probabilitatea ca matricea W aleasă să aibă variația în interiorul clasei este 0. **Metoda Fisherfaces** proiectează setul de imagini înspre rezolvarea acestei problem într-un spațiu minidimensional, astfel matricea de variație în interiorul clasei este diferită de 0.

Wopt, fiind definit ca:



**Procesul de recunoaștere facial folosind algoritmul *Fisherfaces:***

1. Se încarcă setul de date (numit și set de antrenare )
2. Se extrace media locală, medie pentru fiecare clasă în parte.
3. Se extrage media globală a tuturor claselor din setul de date
4. Se determină SB și SW.
5. Se extrag vectorii proprii ( fisherfaces ) din matricea rezultată din înmulțirea matricei inverse SW și SB
6. Se determină proiecția tuturor imaginilor pe vectorii proprii obținuți.
7. Se ia o imagine de test și se determină vectorii proprii pentru compararea cu proiecțiile celorlalte imagini din setul de antrenare.



Figure 1.8.Exemplu set de antrenare



Figure 1.9. Vectori proprii extrași din setul de antrenare din figura 1.8

## Tema propriu-zisă

Tema aplicației See Now este de a recunoaște o față într-o imagine și a oferii anumite informații referitoare la persoana respectivă. Recunoașterea facial va fi realizată folosind algoritmul Fisherfaces și LBPH, combinând rezultatele celor doi algoritmi, mai exact gradul de confidență al acestora, se va estima acuratețea recunoașterii, mai multe detalii vor fi prezentate în capitlul 3.2.3.

În momentul înregistrării, utilizatorul va oferi un set de date despre el, care vor fi ulterior stocate în baza de date, iar pozele oferite de către acesta vor fi folosite ca și set de antrenare pentru baza de cunoaștere.

Aplicația oferă posibilitatea de a trimite o poză către o anumită persoană în cazul situației în care fața din poză este cunoscută sau aplicația va oferi răspuns în ceea ce privește fața din poză, utilizatorul având posibilitatea de a oferi un calificativ evaluării. ( vezi capitolul 3.2.4 )

Toate imaginile în care utilizatorul X a fost identificat sau le-a încărcat pe server vor fi prezente pe profilul acestuia, având posibilitatea de a le descărca pe telefon în orice moment. Comunicația între aplicație și web server este descrisă în capitolul 2.2.2, precum și toate funcționalitățile pe care le oferă serverul aplicației.

Interacțiunile dintre utilizatori se pot descrie sub forma unui graf în care nodurile sunt utilzatorii, iar arcele reprezintă relația de prietenie dintre aceștia, doi utilizatori sunt considerați prieteni dacă aceștia impart o imagine ( apare pe profilul amândurora ).

**Exemplu de use caseuri al aplicației See Now:**

*Eu ca și student, în primele săptămâni am avut dificultăți în a-mi reține numele colegilor, însă eram nevoit să le adresez anumite întrebări, cee ace m-a pus într-o situație foarte incomfortabilă, așadar aplicația See Now pe baza unei imagini cu fața colegului mi-ar putea indica numele acestuia.*

*Totodată prin intermediul aplicației See Now îi pot cere ajutor unui coleg, pe care-l îl cunosc însă am dificultăți în a purta o discuție sau a îniția una, el fiind notificat de către fotografia efectuată, iar în descrierea imaginii fiind întrebarea mea.*

*Memorie insuficientă în telefonul mobil, stochez pozele mele pe profilul meu, le marchez ca și private, voi avea numai eu acces la ele.*

# Medii și tehnologii de programare folosite

## Android Studio



Figure 2.1 Logo-ul Android Studio

**Android Studio** este mediul de dezvoltare integrat ( engl Integrated Development Environment, IDE ) pentru dezvoltarea aplicaților Android, care aduce împreună Java și SDK-ul Android pentru a face simplă scrierea de aplicații Android, el fiind disponibil pentru Windows, macOS și Linux. **Android SDK**, inclus în Android Studio, oferă acces la librăriile Android folosite pentru dezvoltarea aplicațiilor. Editorul de cod suportă IntelliSense ( auto completare ), precum și refactorizare de cod.

**Android Studio** oferă și un editor de layout bogat, care permite utilizatorilor să modeleze componentele UI după bunul plac, în plus vine și cu opțiunea de a previzualiza laytout-urile pe mai multe configurații de ecran. Android Studio are integrat și un dispozitiv Android Virtual ( emulator ) pentru a rula și depana aplicații.

## MySQL Server



Figure 2.2 Logo-ul MySQL

MySQL este soluția software de gestionare a bazelor de date care mă ajută să preiau și stochez date în tabele. Bazele de date sunt relaționale, mai exact datele sunt stocate în tabele separate.

MySQL Server poate rula confortabil pe un desktop, laptop împreună cu alte aplicații, una din proprietățile server-ului fiind viteza acestuia. Totodată MySQL este software open source, ceea ce înseamnă că oricine poate utiliza și modifica software-ul din spatele acestuia.

## OpenCV 3.0

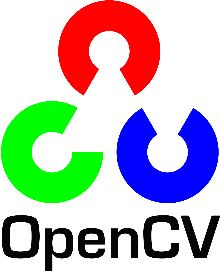


Figure 2.3 Logo-ul OpenCV

OpenCV este o bibliotecă de viziune open source pentru calculatoare. Biblioteca este cross-platform și rulează pe Windows, Linux, Mac OS X, Android și iOS mobil. Biblioteca conține peste 2500 de algoritmi, printer care se alfă și algoritmii prezentați anterior: LBPH, FisherFaces.

Alogritmii implementați în OpenCV pot fi folosiți pentru a detecta și recunoaște fețe, identifica obiecte, a clasifica acțiuni umane în videoclipuri, extracția modelelor 3D, găsirea de imagini similare dintr-o bază de date, urmărirea mișcărilor ochilor, robotică, etc.

Librăria vine cu support pentru **C++, Python, Java** și **Matlab**, initial fiind scrisă în C++.

## Ubuntu Server

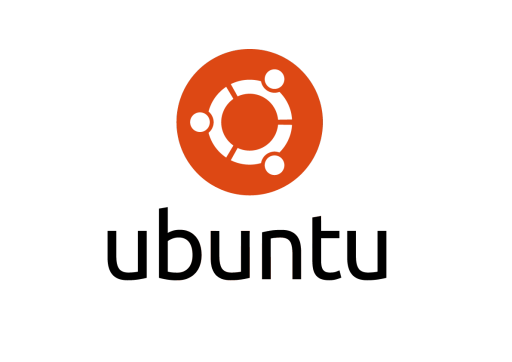


Figure 2.4 Logo-ul Ubuntu

Ubuntu este un sistem de operare open source, gratis și totodată o distribuție Linux bazată pe **Debian.** Ubuntu vine sub 3 varinte: **Ubuntu Desktop, Ubuntu Server, Ubuntu Cloud**.

**Ubuntu Server** rulează pe toate arhitecturile majore: x86, x86064, ARM v7, ARM64, POWER8 și IBM System z, în timp ce SPARC nu mai este oficial suportat de către Ubuntu. Cerințele de sistem minime pentru acesta sunt: 512 MB RAM, 1 GHz CPU și 1 GB spațiu de disc.

Ubuntu Server este folosit pentru partea de Server care va fi descrisă ulterior în **capitlul 3.2.2.**

## Microsoft Visual Studio

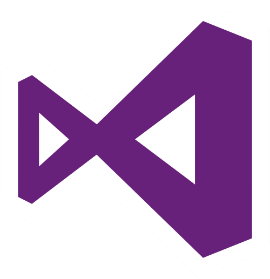


Figure 2.5 Logo-ul Visual Studio

**Visual Studio** este un set complet de instrumente de dezvoltare software produs de Microsfot. Acesta include un editor de cod care suportă IntelliSense ( auto completare ), precum în cazul lui **Android Studio** prezentat anterior, precum și refactorizare de cod.

Visual Studio suportă peste 36 de limbaje de programare diferite, inclusive mod de depanare pentru acestea printer care: C, C++, Visual Basic, C#, JavaScript, Python, PHP.

În lucrarea prezentă Visual Studio a fost folosit pentru scrierea codului PHP care rulează pe Server și Python pentru partea de recunoaștere facială, vezi **capitolul 3.2.3**

## Apache 2

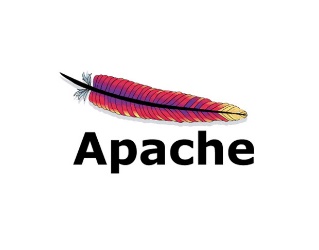


Figure 2.6 Logo-ul Apache

Apache este cel mai frecvent utlizat web server pe sistemele Linux. Web server-ul are rolul de a oferi paginile web cerute de către diferiți clienți ( în cazul nostrum aplicația See Now care va rula pe Android ).

Cel mai comun protocol folosit pentru transferal de date web este **HTTP –** Hyper Text Transfer Protocol, iar pentru încărcarea și descărcarea de fișiere se folosește protocolul FTP – File Transfer protocol, ambele fiind suportate de către Apache 2.

Pentru a instala Apache 2 pe Linux se execută comanda:

sudo apt install apache2

Web serverele care folosesc Apache vin cel mai des în combinație cu un server MySQL și cu un preprocesor PHP sau alt limbaje de programare specifice precum Perl, Python, această configurație este întâlnită sub denumirea de LAMP (Linux, Apache, MySQL, Perl/Python/ Php). Configurația web server-ului utilizat în lucrarea prezentă este descris în capitolul 3.2.2

## Github

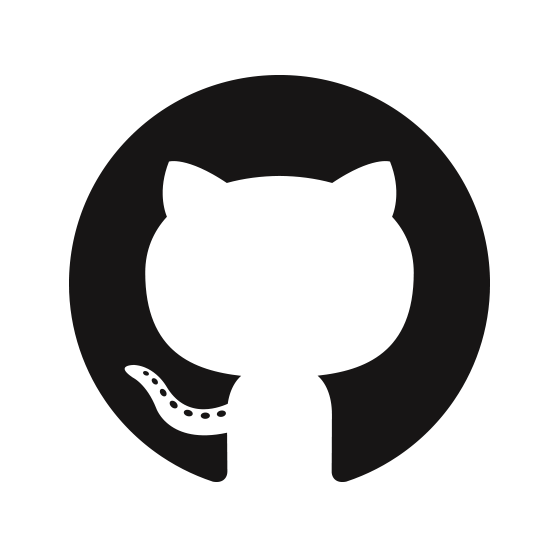


Figure 2.7 Logo-ul GitHub

**Github** este un serviciu web de găzdure pentru versionarea proiectelor folosing git. Oferă suport pentru configuration management, cât și pentru change managment-ul proiectului.

**Git** este un sistem de versionare pentru urmărirea modificărilor asupra fișierelor și coordonarea muncii asupra fișierelor.

**Aplicația descrisă în această lucrare, See Now, folosește 2 repositories, care pot fi accesate liber de către oricine la adresele:**

[**https://github.com/BorduzRazvan/seenow**](https://github.com/BorduzRazvan/seenow) **-** este descriesă aplicația Android – Interfața Utilizator

[**https://github.com/BorduzRazvan/seenow\_server**](https://github.com/BorduzRazvan/seenow_server) - este descris Web Serverul aplicației.

În data de 29/05/2018 pe repository-ul aplicației Android au fost efectuate un număr de 28 commit-rui, de către un utilizator și nici o livrare nu a fost marcată. (vezi figura 2.8), respectiv 4 commit-uri pe repository-ul server-ului.



Figure 2.8 Statistici GitHub: Aplicație SeeNow



Figure 2.9 Statistici GitHub: Server SeeNow

## 2.8 Descrierea limbajelor de programare

Un limbaj de programare reprezintă o colecție de expresii, reguli de formulare a unor instrucțiuni pentru un computer, un limbaj de programare are definit un set de reguli sintactice și semantice, programatorul fiind cel care specific în mod concret acțiunile care urmează a fi executate de către calculator.

### 2.8.1 JAVA

Java este un limbaj de programare orientat pe obiecte, bazat pe clase și concurență. Aplicațiile Java sunt de regulă compilate în bytecodes care va rula pe orice mașină virtuală JAVA ( JVM ) indiferent de arhitectura pe care va rula.

Elementele de bază ale limbajului de programare Java sunt [13]:

* **Simplitate**: Java a fost conceput ca o soluție ușor de înțeles și folosit de către programator.
* **Orientat pe obiecte**: Java nu a fost conceput ca și compatibil la nivel de cod sursă cu alte limbaje de programare, acest lucru a permis dezvoltarea liberă a limbajului Java, acest lucru a dus la o manieră clară, ușor de înțeles și utilizat a obiectelor.
* **Robust**: Pentru a obține fiabilitate, Java vă limitează în câteva domenii cheie pentru a vă forța să vă găsiți greșelile devreme în dezvoltarea programului.
* **Interpretabil și dinamic**: Programele Java transporta cu ele cantități substanțiale de informații de tip run-time care sunt folosite pentru a verifica și rezolva accesul la obiecte la momentul executării. Acest lucru face posibilă conectarea dinamică a codului într-un mod sigur și eficient.
* **Portabil**: Același cod trebuie să funcționeze pe toate computerele. Prin urmare, au fost necesare unele metode de generare a codului executabil portabil.

Limbajul de programare Java este cheie în dezvoltarea aplicațiilor Android, deși acesta este construit pe nucleul Linux, scris în mare parte în C, Android SDK-ul folosește limbajul **Java.** Însă bytecodes generați în urma compilării sunt incompatibili cu **mașina virtuală Java**, în cazul aplicațiilor Android fiind interpretat de către mașina virtuală **Davlik**.

În 2016 Java a fost declarant cel mai popular limbaj de programare în uz.

### Python & OpenCV

Python este un limbaj de programare interpretat la nivel înalt pentru programare generală. Python este menit să fie un limbaj ușor de citit, utilizând adesea cuvinte cheie în limba engleză unde alte limbaje de programare folosesc semen de punctuație. Spre deosebire de alte limbaje de programare nu se folosesc paranteze curbate pentru a delimita blocurile, iar punct și virgule după declarații este opțională, având astfel mai puține excepții sintactice și cazuri special în comparație cu C sau Java. [14]

Acesta vine cu o mare bibliotecă standard care acoperă domenii cum ar fi procesarea șirurilor (expresii regulate, Unicode, etc. ), protocoale Internet ( HTTP, FTP, etc. ),și interfețele sistemului de operare ( apeluri sistem, sisteme de fișiere ). Totodată biblioteca OpenCV ( vezi capitolul 2.3 ) este disponibilă în limbaj Python, implementarea acesteia fiind o extensie a implementarii originale OpenCV C++.

Un dezavantaj al limbajului de programare Python fiind însă viteza de execuție a acestuia.

### PHP

PHP (acronim pentru Hypertext Preprocessor) este un limbaj de programare, open source, utilizat pe scară largă, care este potrivit în mod special pentru dezvoltarea aplicațiilor web și poate fi integrat în HTML.

Diferența între PHP și JavaScript este reprezentată de faptul că codul este executat pe server, generând HTML care apoi este trimis către client.

PHP este un limbaj de programare simplu de perceput de către programator, cu o multitudine de facilități avansate, deasemenea acesta poate fi utilizat pe majoritatea sistemelor de operare: Linux, Microsoft Windows, Unix, Mac OS X, RISC OS.

Scripturile PHP sunt utilizate în trei domenii principale:

- **Scripting de partea serverului**, acesta reprezentând domeniul de bază PHP. Rezultatele programului PHP pot fi accesate din navigatorul web, vizualizând pagina prin server.

- **Scripting în linia de comandă,** PHP poate să ruleze fără ajutorul unui server sau navigator web, acest mod de utilizare fiind utilizat pentru executarea scripturilor în mod regulat pe Linux cu cron, respectiv pe Windows cu Task Scheduler.

- **Aplicații de partea clinetului.** PHP nu este cel mai potrivit limbaj pentru a creea aplicații pentru partea de client, însă pentru cunoscătorii limbajlui există facilități avansate ale PHP-ului pentru a realiza interfețe grafice, precum PHP-GTK.

Pe lângă cele prezentate anterior PHP realizează conexiunea între o bază de date și aplicație într-un mod foarte ușor de înțeles de către programator, totodată PHP cunoaște următoarele protocoale: LDAP, IMAP, SNMP, POP3, HTTP, COM, etc.

Pentru a declanșa comenzile PHP, trebuie să înveți o nouă etichetă. Iată prima parte:

**<? Php**

Primul lucru pe care îl puteți observa este că eticheta nu a fost închisă. Asta pentru ca întreaga secțiune a PHP poate fi plasată în interiorul acestei etichete și se termină numai atunci când se întâlnește o parte de închidere, care arată astfel:

**?>**

### SQL

**SQL** este un limbaj de interogare structurat, specific pentru manipularea datelor in sistemele de manipulare a bazelor de date relationale (RDBMS). Principalele acțiuni ale effectuate fiind: inserțiia, interogarea, actualizarea, ștergerea, modificarea și crearea tabelelor, etc.

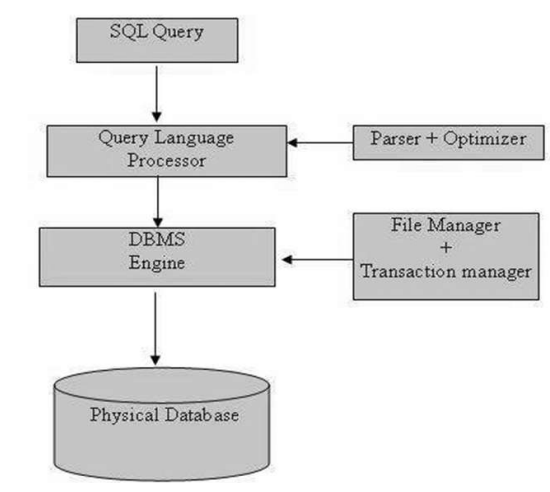


Figure 3 Arhitectura SQL[17]

**Tipuri de date în SQL:**

* **CHARACTER** ( sau CHAR ) – șir de caractere
* **INTEGER** ( sau SMALLINT ) – număr întreg
* **FLOAT, REAL** - număr real
* **NUMERIC**(precision, scale) - unde precision reprezintă numărul de cifre din partea întreagă, iar scale reprezintă numărul de zecimale.
* **DATE** - data zilei
* **TIME** - ora

Instrucțiuni de bază SQL

|  |  |
| --- | --- |
| **Comandă** | **Descriere** |
| CREATE | Crează o nouă tebelă sau un alt obiect în baza de date |
| ALTER | Modifică un obiect al bazei de date existent, exemplu: un tabel. |
| DROP | Șterge un table |
| SELECT | Primește informații cu din unul sau mai multe tabele |
| INSERT | Inserează o înregistrare în tabelă |
| UPDATE | Modifică o înregistrare existent |
| DELETE | Șterge o înregistrare |
| GRANT | Oferă privilegii unui utilizator |
| REVOKE | Șterge privilegile unui utilizator |

### C++

C++ este un limbaj de programare cu scop general, are funcții de programare orientate pe obiecte și generice, însă tot odată oferă și facilități pentru manipularea memoriei, așadar suportă programarea la nivel scăzut ( low-level ).

Limbajul de programare C++ este prezent în lucrarea scrisă prin implementarea algoritmilor pentru detecția și recunoașterea facială din OpenCV, vezi capitolul **3.2.3.**

Avantajele limbajului de programare C++: este orientat pe obiecte, este portabil, limbaj de programare la nivel scăzut (engl. low level language ), este compatibil cu limbajul C, folosește polimorfismul și moștenirea, folosește reutilizarea codului, este rapid.

Însă acesta prezintă și anumite dezavantaje: nivel de securitate scăzut, dependent de platform ape care va rula, nu suportă multithreading.

Exemple de aplicații care folosesc C++: Adobe Photoshop, Amazon, Facebook.

# Documentație tehnică a aplicației

Acest capitol are ca scop descrierea aplicației: design și implementare. Aplicația See Now prezentată anterior poate fi divizată în patru mari părți: s**erver, baze de date, interfață utilizator și altoritmi de detecție și recunoaștere facială.**

Specificațiile funcționale ale sistemului care urmează a fi descris (engl. Designed ) și implementat sunt:

1. Utilizatorul are posibilitatea de a-și creea un cont nou sau a se autentifica folosind un cont social ( ex: cont Facebook ).
2. Utilizatorul are posibilitatea de a își recupera parola în cazul în care acesta a uitat-o.
3. Utilizatorul are posibilitatea de a își schimba adresa de email sau parola.
4. În momentul înregistrării normale utilizatorul trebuie să ofere următoarele informații pentru a finaliza procesul de înregistrare: **nume, prenume, email, parola, data nașterii, sex, țara de origine, o poză cu fața lui.**
5. În cazul autentificării folosind un cont social ( ex: cont Facebook ), trebuie obținute următoarele informații: **nume, prenume, email, data nașterii, sex, țara de origine, pozele utilizatorului.**
6. Toate pozele încărcate se vor stoca pe server.
7. În orice moment utilizatorul are acces la pozele lui și dispune de o metodă de a le șterge sau a le face private ( astfel încât nuami el să le vadă, nu și prietenii lui. ).
8. O poză poate fi impoărțită între mai mulți utilizatori.
9. Aplicația trebuie să suporte următoarele acțiuni:
   1. Încarcare imagine
   2. Încarcare față -> cunoscută: Utilizatorul are posibilitatea de a încărca o poză pe server cu o persoanoă pe care o cunoaște, crescând astfel acuratețea predicțiilor viitoare.
   3. Încărcare față -> **necunoscută:** Utilizatorul are dreptul de a întreba serverul cine este persoana dintr-o imagine de **n ori pe zi.**
   4. Vizualizează profil -> **vizualizarea numărului de imagini încărcate** pe server, **numărul de imagini în care apare**, **numărul de prieteni**, **număr de puncte**, dar și **toate imaginile.**
   5. Descărcare imagine.
   6. Descărcare set imagini.
   7. Descărcare toate imagini.
   8. Notificare utilizator în cazul în care acesta a fost identificat în vreo imagine de către un utilizator.

Obs. În cazul unui set de mai multe imagini, acestea se vor arhiva într-un format standard.

1. Utilizatorul are posibilitatea de a se face invizibil, mai exact de a utiliza aplicația decât în scop de stocare imagini, acesta nefiind identificat de alți utilizatori, însă nu va dispune nici el de funcția de recunoaștere persoane.
2. Se consideră doi utilizatori ca și perieteni, acei utilizaori care impart o imagine, adică: Dacă utilizatorul X trimite o poză cu utilizatorul Y, în acel moment ei sunt considerați prieteni. Aceștia sunt considerați prieteni atâta timp cât ambii acceptă acest lucru.
3. Orice utilizator are dreptul de a vizualiza imaginile încărcate de către prietenii acestuia, care sunt marcate ca și publice.
4. Imaginile pot fi appreciate de către utlizatori sau distribuite pe alte platforme de socializare. ( Ex: Facebook ).
5. Pentru fiecare imagine cu o față încărcată de utlizator pe server acesta va fi recompensat cu un punct.

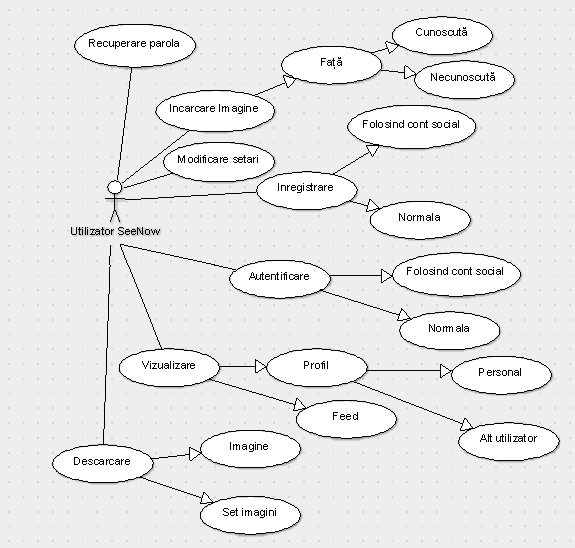
Specificațiile de sistem au ca rol satisfacerea nevoilor unui utilizator al aplicației, nevoi descriese în figura 3.1.

Figure 3.1 Descriere usecase-uri folosind ArgoUML

## Design

Rolul acestui capitol este de a stabili interacțiunea dintre componentele ce alcătuiesc sistemul See Now înspre îndeplinirea specificațiilor funcționale ale acestuia. Din diagram usecase-urilor prezentată anterior se pot deduce următoarele mari capitole funcționale:

1. **Managementul contului** - care are ca rol implementarea funcțiilor de înregistrare, autentificare, recuperare parola, modificare setări. Acest capitol functional are impact asupra: bazei de date, serverului și interfeței utilizator.
2. **Încărcare imagini** – care are ca rol implementarea funcțiilor de încărcare imagini cu sau fără fețe, impact asupra: bazei de date, server și interfeței utilizator.
3. **Vizualizare și descărcare imagini** – care are ca rol implementarea funcțiilor de vizualizare, respective descărcare a imaginilor, impact asupra: bazei de date, server și interfață utilizator.
4. **Recunoaștere persoană** – are ca rol identificarea unei persoane pe baza unei imagini stocate deja pe server, impact asupra: server ( *vezi capitolul 3.2.4. Algoritmi de detecție și recunoaștere facială* ) și interfața utilizator.
5. **Notificare** – are rolul de a notifica un utilizator, impact asupra: server și interfață utilizator

Toate datele care urmează a fi stocate pe server, vor fi introduce în diferite tabele, oferind astfel posibilitatea de a fi interrogate ușor pe viitor. Algoritmii de detecție și recunoaștere facială vor fi antrenați permanenți folosind baza de cunoștințe dobândită până la momentul antrenării, în acest sens cu cât vor exista cât mai multe imagini cu fața unui utilizator, cu atât mai precisă va fi predicția finală.

Aplicația Android va comunica cu serverul folosind protocolul HTTP, iar pe server există baza de date MySQL, scripturi scrise în PHP care vor lansa interogării asupra acesteia și tot odată scripturi .python și .c++ pentru rularea algoritmilor de detecție și recunoaștere facială. Pe lângă comunicația între SERVER și aplicația Android ( CLIENT ), poate exista în anumite situații și o comunicație între CLIENT și FACEBOOK pentru preluarea de date.

Voi folosi formatul JSON ( JavaScript Object Notation ), acesta fiind un standard ușor de înțeles și prelucrat bazat pe text folosit în schimbul de date. În sistemul descris anterior, formatul JSON va fi folosit în următorul fel:

1. Aplicația android ( clientul ), se va conecta la serverul PHP, va codifica informațiile relevante sub formatul JSON și le va transmite către server.
2. Scriptul PHP, care va rula pe server, va prelua datele în format JSON, le va decodifica, va pregăti răspunsul pentru client, pe baza unor acțiuni ( interogări în baza de date, rularea altor scripturi ), va codifica informația în format JSON și o va trimite înapoi clientului.
3. Aplicația android ( clientul ) va prelua datele recepționate ulterior și pe baza lor va lua anumite decizii sau va afișa un anumit conținut.



Figure 3.2 Exemplu arhitectură CLIENT-SERVER

Aplicația android ( clientul ) trebuie să primească permisiuni de la utilizator pentru a efectua următoarele operații:

1. **Scrieri și citri din memorie**
2. **Folosirea internetului**
3. **Folosirea camerei**
4. **Scrierea și citirea din memorie** este necesară deoarece aplicația See Now, va citi imaginile din telefon, selectate de către utilizator pentru a fi încărcate pe server, totodată aplicația va creea un director /SeeNow\_Images unde va stoca toate imaginiile capturate din aplicație ( pot fi șterse manual de către utilizator ).
5. **Folosirea internetului** este necesară pentru comunicația între CLIENT și SERVER, respective CLIENT și FACEBOOK.
6. **Folosirea camerei** este necesară pentru a putea captura imagini folosind camera foto a telefonului direct din aplicația See Now.

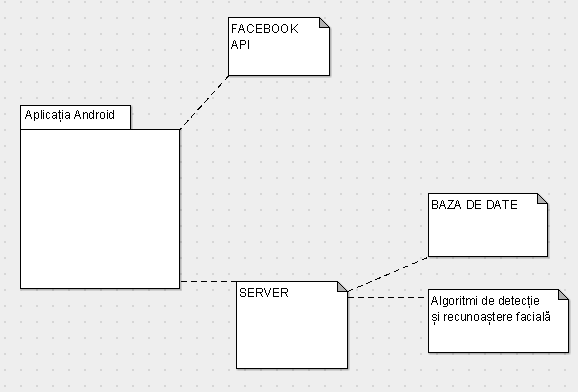


Figure 3.3 Arhitectura sistemului SeeNow

Fiecare specificație de sistem vine cu un impact asupra unei părți din sistem sau asupra mai multor. Acțiunile necesare și effectuate în situațiile resepective vor fi descries în capitolul 3.2 Implementare, unde se va detalia felul în care au fost implementate la nivel software specificațiile de sistem, astfel încât ele să se comporte exact ca în specificații.

Spre deosebire de restul capitolelor, implementarea algoritmilor de detecție și recunoaștere facială vor fi implementați atât în pyhon, cât și în C++, iar în urma testelor efectuate și descrise în capitolul **4.0 Testare**, se decide care dintre cele două implementări este oportună și va fi folosită ulterior.

Totodată în prezenta lucrare va fi descris și felul în care utlizatorul urmează să utilizeze aplicația, aspect prezentat în capitolul **5.0 Manual de utilizare,** fiind descrise toate usecase-urile prezentate anterior.

## Implementare

Acest capitol are ca scop explicarea felului în care au fost implementate specificațiile de sistem la nivelul fiecărei componente, așadar exact cum a fost prezentat anterior, compoenentele sunt: baza de date, server, UI și algoritmi de detecție și recunoaștere facial, care vor fi implementați în MySQL, PHP, Java , Python și C++.

### 3.2.1 Baza de date

Această componentă are ca scop definirea tabelelor în care vor fi stocate toate informațiile necesare, definite relații și oferirea lor către aplicație prin intermediul serverului.

Baza de date a aplicației va purta numele SeenowDB.

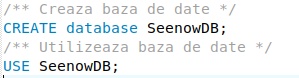


Figure 4 Comenzi de creeare și selectare a bazei de date



Figure 5 Răspuns comenzi prezentate în figura 4.

Tabela **users** care are ca scop reținerea informațiilor esențiale despre un utilizator, informații precum: **nume, prenume, email, parola, data nașterii, sex, țara de origine, o poză cu fața lui.** Informația cu privire la fața lui va veni din altă tabelă, numită pictures, astfel fiind descrie relații de tip **ONE – to – MANY.** Pe lângă informațiile prezentate mai sunt necesare: **activation\_code** – care va fi trimis utilizatorului pe mail pentru a-și active contul, **data\_înregistrării** – momentul când s-a înregistrat utilizatorul, **useRecognizer –** dacă utilizatorul este de acord sau nu să folosească recunoașterea persoanelor oferită de aplicație, **social\_id** – id-ul social al utilizatorului oferit de către API-ul facebook în cazul în care acesta se înregistrează folosind contul său facebook, **points** – numărul de puncta acumulate, **salt –** pentru criptarea parolei, **id –** identficator primar al utilizatorului în baza de date. use

**Se va considera un utilizator înregistrat cu success acel utilizator pentru care câmpul** activation\_code este null. Totodată se consideă emailul ca și un atribut unic al fiecărui utilizator, poate exista în baza de date un singur utilizator cu o anumită adresă de mail.

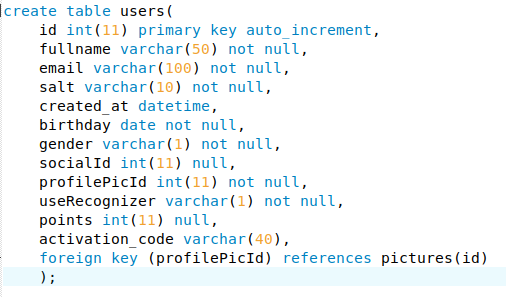


Figure 6 Comanda de creeare a tabelei utilizator

Se poate obesrva relația dintre tabela **users** și **pictures,** așadar apare nevoie creeri unei tabele **pictures** care să conțină **numele imaginii**, **visibilitatea imaginii (** vizibilă, invizibilă ), respective un id.

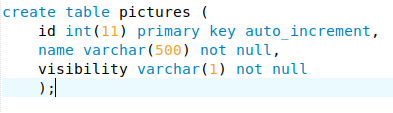


Figure 7 Comanda de creeare a tabelei pictures

Însă acum apare nevoia de a cunoaște cine este autorul unei imagini, mai exact cine a încărcat imaginea pe server, definindu-se astfel tabela: **pictures\_author,** care va conține:**id, pictureId –** id-ul imaginii pe care utilizatorul a încărcat-o, respective **authorId –** id-ul utilizatorului care a încărcat-o.

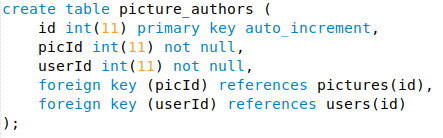


Figure 8 Comanda de creeare a tabelei picture\_authors

Până în acest moment am definit structura tabelelor în care se vor stoca informațiile esențiale despre utilizator.

Pentru inserția în tabela utilizator există două scenarii:

1. Înregistrare normală.
2. Înregistrare folosind cont facebook.
3. Pentru înregistrarea normală câmpul **socialId** va fi null, astfel aplicația va fi notificată de faptul că utilizatorul nu s-a înregistrat cu contul său facebook. Ordinea inserării în tabele este:
   1. Se inserează informațiile privind imaginea încărcată de utilizator pe server.



Figure 9 Exemplu înserție în tabela pictures

Se poate observa în figura 9 faptul că visibility a luat valoarea v – ceea ce va spune serverului faptul că imaginea este vizibilă, dacă era valoarea i în câmpul respective invizibilă– imaginea nu era vizibilă decât de către autor.

* 1. Se inserează informațiile utilizatorului, inclusive id-ul imaginii încărcate.



Figure 10 Selecția id-ului imaginii inserate în tabela pictures în figura 9 

Figure 11 Exemplu inserție în tabela utilizator fără socialId

Se poate observa în figura 10 faptul că am inserat în câmpul gender (sexul utilizatorului) valoarea m ceea ce înseamnă că este sex masculine, cealaltă valoare acceptată fiind f pentru feminine. Deasemenea un alt aspect care se poate observa este valoarea inserată în câmpul useRecognizer – d – de la DA, cealaltă valoare fiind n – de la NU.

1. Pentru înregistrarea folosind cont facebook, informațiile vor fi oferite de către aplicație, însă în plus se va actualize și câmpul socialId, iar câmpul cod\_activare va fi null în acest caz nefiind necesară nici o activare a contului, iar ordinea fiind aceeași, însă diferit va fi felul de inserție definit la pasul b. 

Figure 12 Exemplu inserție în tabela utilizator folosind socialId

Pe lâgnă operația de inserție în tabela **users** mai sunt necesare definirea comenzilor:

**Autentificare**

* Selectăm din tabela users acea înregistrare unde adresa de **email** coincide cu cea introdusă de utilizator. Verificarea dacă contul este activat sau corectitudinea parolei introduse fiind făcută de către server.

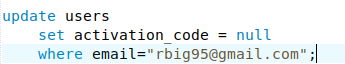


* Selectăm din tabela users acea înregistrare unde **socialId** coincide cu cel oferit de către API-ul de facebook pentru a vedea dacă utilizatorul s-a mai autentificat folosind acel cont de facebook.

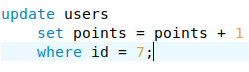


**Actualizare informații tabelă**

* Ștergere **cod de activare** pentru un anumit utilizator pe baza adresei de **email**, care indică faptul că utilizatorul și-a activat contul.



* Incrementare **număr** **puncte** pentru un utilizator al cărui id îl avem.



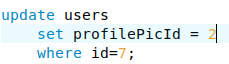
* **Modificare** adresă **email** pentru un utilizator al cărui id îl avem , în momentul modificării adresei de email se va retrimite un cod de confirmare pe acea adresă de email, în tot acel timp contul devenind neutilizabil.



* **Modificare** parolă pentru un utilizator al cărui id îl avem.



* Setare **poză de profil nouă**, important fotografia de profil trebuie să fie aleasă din cele existente deja în tabela pictures.



**Resetare parolă**

Pentru implementarea acestei funcționalități se va folosi tot câmpul activation\_code care în va avea următoarea formă:

**Un\_code.X** –unde Un\_code reprezintă un șir de caractere care reprezintă codul propriu zis, iar .X reprezintă rolul codului: P- resetare parolă, A- activare cont, E- modificare adresă email.

Ex: asddfdfsgfsgfdgfdgfd.A – este un cod de activare

Dsadasjdkslfjdsaklfdsa.P – este un cod de resetare parolă.

Evaluarea valorii din câmpul activation\_code fiind efectuată de către **server.** Toate aceste comenzi vor fi lansate de către server.

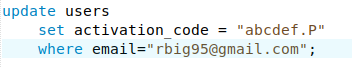


Figure 13 Setare activation\_code pentru resetare parolă

Pe lângă tabelele deja prezentate: users, pictures, picture\_author mai apare necesitatea definirea următoarelor tabele: **feeds, feedLiked**, **usersRelations.**

Tabela feeds are rolul de a oferi informații referitoare la informațiile ce vor fi afișate în aplicație și va conține următoarele informații: autorul fotografiei, utilizatorul identificat în imagine dacă este cazul, id-ul fotografiei, descriere, data postării și un nivel de încredere al recunoașterii dacă este cazul.

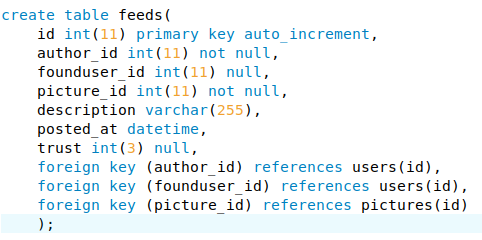


Figure 14 Comanda de creeare a tabelei feeds

Se defines două tipuri de inserție în tabela **feeds,** una când în imagine este identificat un utilizator, respectiv când nu este identificat:



Figure 15 Exemplu 1 Inserție în tabela feeds

În figura 15 se inserează în tabela feeds o înregistrare din care aflăm faptul că userul cu id-ul 7 a încărcat pe server o imagine cu id-ul 1 în care se găsește userul cu id-ul 11 și descrierea imaginii este Prima imagine.



Figure 16 Exemplu 2 inserție în tabela feeds

În figura 16 se inserează în tabela feeds o înregistrare din care aflăm faptul că userul cu id-ul 7 a încărcat o imagine pe server în care nu s-a identificat nici un utlizator existent în baza noastră de date, descrierea fiind ” Cine e oare? ”.

Tabela **feedLiked** are ca rol stabilirea relațiilor de tip One-To-Many între tabela users și tabela feeds, astfel încât un utlizator poate aprecia mai multe elemente ale feed-ului. Așadar tabela feedLiked, conține: id – identificator unic al intrării din tabelă, feed\_id – identificatorul înregistrării din tabela feeds care este apreciată, respective user\_id – identificatorul utilizatorului din tabela users care apreciază feed-ul.

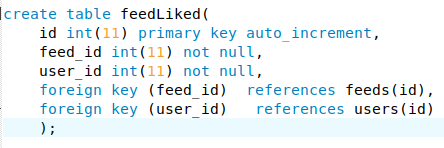
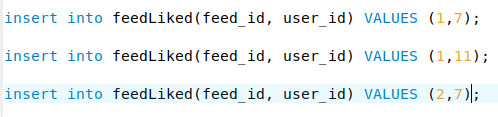


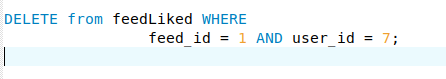
Figure 17 Comanda de creeare a tabelei feedLiked

Pentru tabela feedLiked sunt necesare următoarele comenzi:

* Adăugare element în tabelă în urma acțiunii de apreciere



* Eliminare element din tabelă în urmă eliminării aprecierii.



* Număr total de aprecieri pentru un anumit feed.



Tabela **usersRelations** reține interacțiunile dintre utilizatori, mai exact dacă utilizatorul X încarcă o imagine în care apare și utilizatorul Y în acel moment se definește automat o relație între cei doi, care va fi prezentă atâta timp cât imaginea există în tabela feeds, așadar acest table va avea un id, user1\_id, respective user2\_id, în momentuil în care se va define o relație între user1\_id și user2\_id, automat se va define și între user2 și user1.

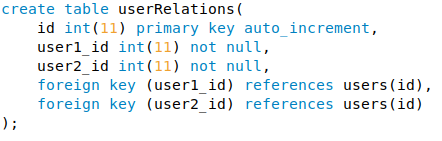


Figure 18 Comanda de creeare a tabelei userRelations

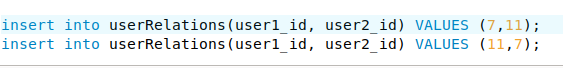


Figure 19 Inserarea unei relații între2 utilizatori

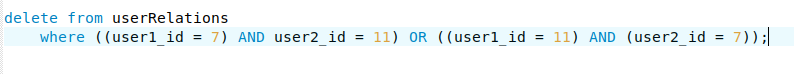
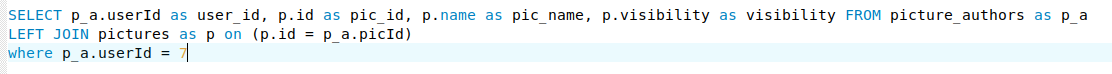


Figure 20 Ștergerea relației dintre 2 utilizatori

Acum că am definit toate tabelele necesare aplicației, mai trebuie definite anumite comenzi specifice:

* Obținerea tuturor imaginilor pentru un anumit utilizator.



* Obținerea numărului de utilizatori cu care un utilizator are definită o relație.



* Obținerea informațiilor care urmează a fii afișate pe feed pentru un utilizator ( toate elementele din feed în care este implicat utilizatorul sau un alt utilizator cu care acesta se află într-o relație ).

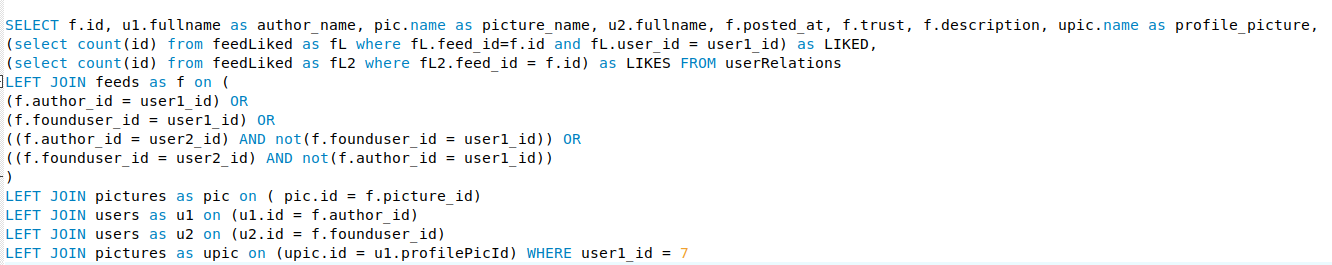


Figure 21 Informatii pentru feed-ul utilizatorului cu id = 7

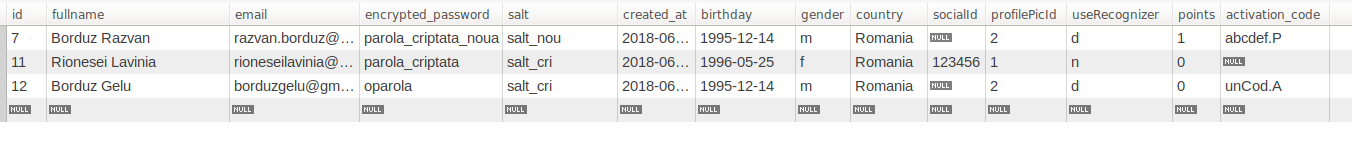
Selecția având loc pe baza relațiilor dintre utilizatori:

* Se aleg toți utilizatorii cu care utilizatorul cu id 7 are definită o relație
* Se aleg toate alementele din tabela feeds pentru care:
  + Autorul sau utilizatorul găsit este utilizatorul cu id 7
  + Autorul sau utilizatorul găsit este un utilizator care are definită o relație cu utilizatorul cu id 7 și în plus utilizatorul cu id 7 nu este celălalt utilizator din feed.
* Se alege elemental din tabela pictues care are id-ul precum cel din elementul selectat din feeds anterior.
* Se aleg utilizatorii u1 și u2 – u1 fiind autorul, u2 fiind utilizatorul găsit
* Se afișează: numele utilizatorilor, numele imaginii, data postării, nivelul încrederii în recunoașterea efectuată, descrierea, LIKE (dacă a dat sau nu utilizatorul cu id-ul 7 like), LIKES numărul de like-uri.

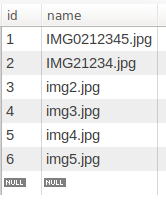
Exemplu:

Pentru următoarea configurație:

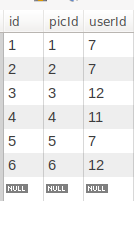
* Utilizatori ( select \* from users )

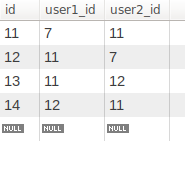


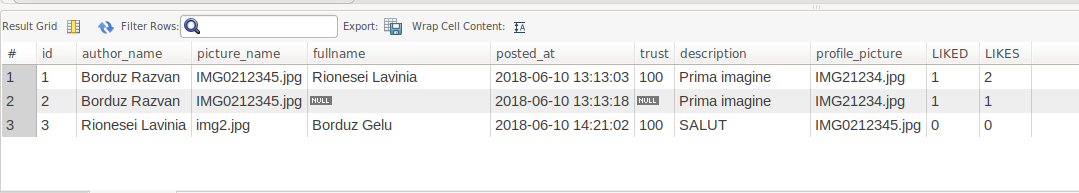
* Imagini ( select \* from pictures )



* Autori imagini ( select \* from picture\_authors ).

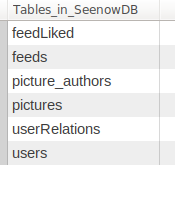


* Relații între utilizatori ( select \* from userRelations )
* 
* Elemente din tabela feed ( select \* from feeds)



Răspunsul la comanda prezentată anterior ( vezi figura 21 )

Pe scurt în acest capitol am definit felul în care este stocată informația despre un utilzator. În acest process sunt implicate mai multe tabele:



Urmând ca prelucrarea lor să fie descrisă în capitolul de implementare al serverului.

### 3.2.2 Server

Serverul are ca scop prelucrarea datelor stocate în baza de date, oferirea de informații către clienți și executarea algoritmilor de detecție și recunoaștere facială.

Așadar implementarea serverului poate fi împărțită în trei părți: prelucrarea bazei de date, stocare și descărcare imagini, respective lansarea algoritmilor de detecție și recunoaștere facială.

**Se instalează apache2 folosind comanda:**

***apt-get -y install apache2***

**Se instalează MySQL Server folosind comenzile:**

*apt-get -y install mysql-server mysql-client*

*mysql\_secure\_installation*

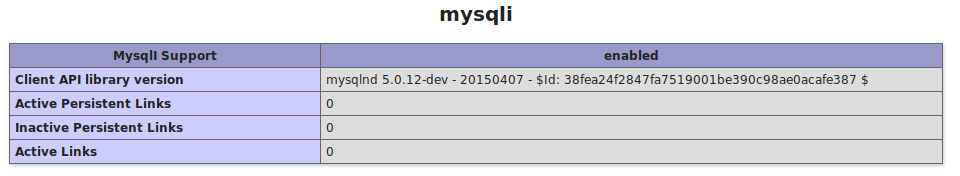
**Se instalează PHP folosind comanda:**

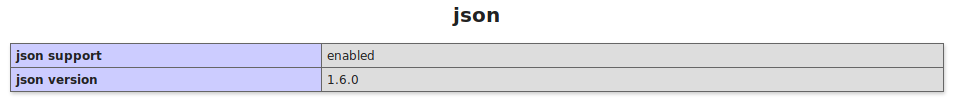
***apt-get -y install php7.2 libapache2-mod-php7.2***

**Se instalează suportul MySQL pentru PHP**

*apt-get -y install php7.2-mysql php7.2-curl php7.2-gd php7.2-intl php-pear php-imagick php7.2-imap php-memcache  php7.2-pspell php7.2-recode php7.2-sqlite3 php7.2-tidy php7.2-xmlrpc php7.2-xsl php7.2-mbstring php-gettext*

În urma rulării phpinfo() se poate observa rezultatul instalării și totodată configurația instalata, unde se pot observa următoarele:



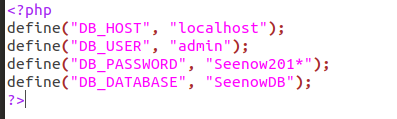


1. **Prelucrarea bazei de date**

Pentru a putea face acest lucru primul pas este definirea conexiunii cu baza de date.

În folderul **/include/** se vor afla toate fișierele de configurare php și scripturi php care vor fi apelate ulterior în funcție de cererile venite de la client.

Definesc fișierul **config.php,** care va conține macro-uri folosite pentru conectarea la baza de date din PHP folosind mysqli.



În fișierul **db\_connect.php**, definesc clasa DB\_Connect, în care pe baza informațiilor din fișierul **config.php** va stabili conexiunea cu baza de date, returnând în final obiectul rezultat în urma conexiunii, el fiind folosit ulterior pentru a executa comenzi SQL asupra bazei de date.



Până în acest moment am definit felul în care mă conectez din PHP la baza de date a aplicației, urmează să definesc clasa DB\_Functions în interiorul fișierului **DB\_Functions.php** care va definii metode pentru fiecare comandă care urmează a fi executată asupra bazei de date ( vezi capitlul 3.3.1 ). În constructorul ei se stabilește conexiunea între php și baza de date.

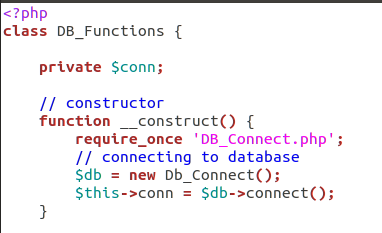
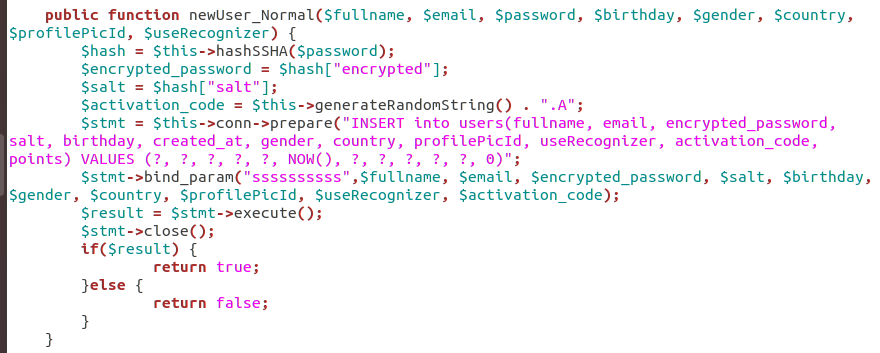


Figure 22 Constructor clasă DB\_Functions

Acum voi define câte o funcție pentru fiecare comandă prezentată în capitolul 3.3.1.

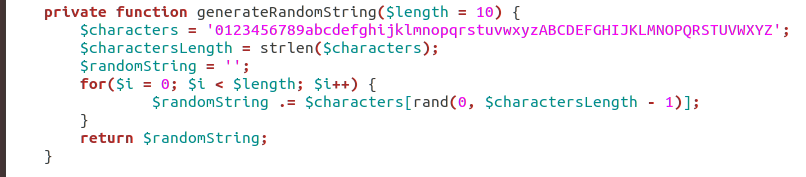
**Comanda înregistrare normală utilizator**



Unde metoda hashSSHA are ca scop criptarea parolei, folosind SHA1 – Secure Hash Alghoritm, iar metoda returnează un atât parola criptată sub eticheta encrypted, cât și valoa folosită pentru criptare sub eticheta salt.



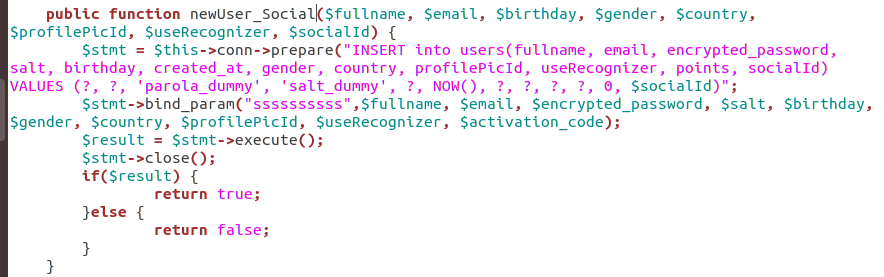
Metoda genera generateRandomString() are ca scop generarea unui șir de caractere de lungime 10, care va fi ulterior folosit ca și cod de activare, fiind concatenate cu .A – ceea ce înseamnă că rolul lui este pentru activare.



Rezultatul metodei newUser\_Normal fiind TRUE în cazul în care utilizatorul a fost înregistrat cu success, FALSE în caz de eroare.

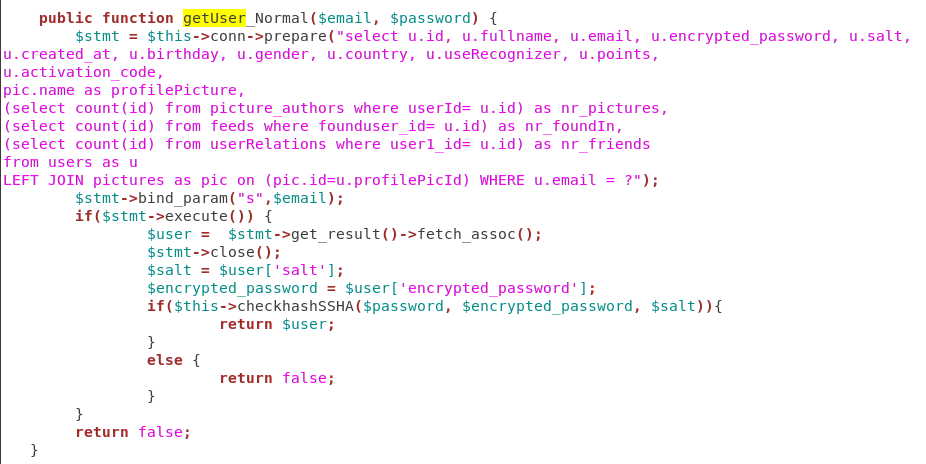
**Comanda înregistrare utilizator folosind rețea socială**

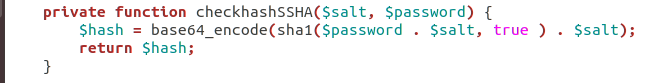
Este similară cu cea pentru înregistrarea normal a utilizatorului, diferă însă faptul că nu este necesar codul de activare și este setat și atributul socialId, iar pentru parolă și salt este folosită o valoare dummy, nefiind necesare.



**Comanda autentificare utilizator pe baza adresei de email și a parolei**

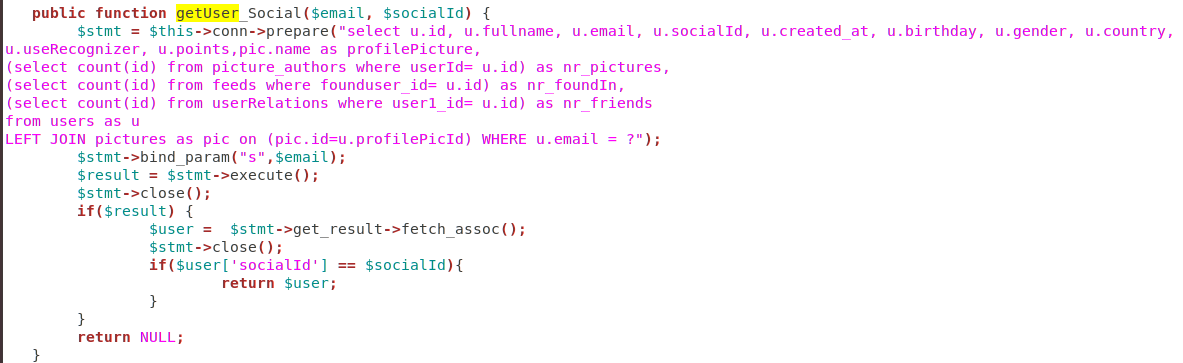
* Se selectează din tabela users înregistrarea care conține adresa de email oferită ca și parametru.
* Se criptează parola oferită ca și parametru folosind valoarea salt din instanța rezultată în urma selecției folosind metoda checkSSHA.
* Se verifică dacă parola criptată din tabelă este identică cu parola oferită ca și parametru criptată apoi. Dacă sunt egale se returnează instanța citită din tabel, altfel se returnează null.





**Comanda autentificare utilizator pe baza adresei de email și a socialId-ului**

În cazul autentificării folosind socialId-ul se selectează din tabel pe baza adresei de email, iar dacă instanța rezultat are același socialId ca și cel oferit ca parametru se returnează instanța selectată, altfel se returnează NULL.



**Comanda resetare parolă**



Se actualizează valoarea activation\_code-ului cu un cod generat care se va termina în .P indicând faptul că scopul este pentru a confirma resetarea parolei.

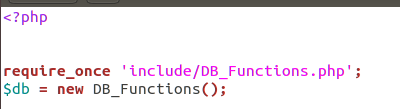
Similar cu metodele prezentate au fost definite pe baza comenzilor din capitolul 3.3.1 următoarele metode:

* + insertImage($name, $visibility, $userId) – inserează o imagine atât în tabela pictures, cât și definește relația dintre ea și autor în tabela picture\_authors
  + setNull\_ActivationCode($id, $activation\_code) – setează activation\_code pentru un utilizatorul al cărui id se cunoaște dacă și numai dacă activation\_codeul oferit ca parametru coincide cu cel din tabelă.
  + getPictures($id) – se selectează toate numele de imagini pentru un utilizator al cărui id se cunoaște.
  + addPoints($id) – adauga 1 punct pentru un utilizator al carui id se cunoaste
  + setEmailAdress($email, $id) – setează adresa de email pentru un utilizator al cărui id se cunoaște, se va seta și activation\_code specific pentru confirmarea noi adrese de email.
  + setPassword($password, $id) – setează o nouă parolă pentru un utilizator al cărui id se cunoaște, se va cripta noua parolă și se va stoca atât parola criptată cât și valoarea cu care a fost criptată parola.
  + setProfilePic($id, $profilePicId) – setează o nouă fotografie de profil pentru un utilizator
  + addFeed\_Known($author\_id, $founduser\_id, $picture\_id, $description, $trust) – inserează o instanță în tabela feeds pentru o imagine în care este recunoscut un utilizator
  + addFeed\_Known($author\_id, $picture\_id, $description) – inserează o instanță în tabela feeds pentru o imagine în care este nu recunoscut un utilizator
  + addLike($feed\_id, $user\_id) – adaugă în tabela feedLiked faptul că utilizatorul cu id-ul $user\_id a apreciat feed-ul cu id-ul $feed\_id
  + removeLike($feed\_id, $user\_id) – șterge înregistrarea din tabelă.
  + addRelation($user1\_id, $user2\_id ) – definește o relație între 2 utilizatori.
  + deleteRelation($user1\_id, $user2\_id) – șterge relația dintre 2 utilizatori.
  + getFeeds($id) – obține toate informațiile pentru un utilizator al cărui id se cunoaște care urmează a fii afișate în feed.

1. **Operații client- server**

Acum că au fost definite majoritatea funcțiilor care vor acționa asupra bazei de date, voi definii următoarele tipuri de cereri din partea clientului ( aplicația Android ): **login**, **register, feed, upload, recognize, gallery, activation** și **download, actions**

Toate fișierele .php care vor fi descrise ulterior vor avea în antet incluse fișierul în care au fost definite metodele anteriore, iar răspunsul serverului către client va fi sub forma de JSON.



1. **Login.php** – este fișierul care va analiza cererea de autentificare a clientului.

Cererea validării autentificării normale:



* Dacă codul de activare nu este null se va returna mesaj de eroare: ACTIVATION\_CODE\_IS\_SET.
* Intrările vor fi analizate la nivel de aplicație.

Cererea validării autentificării Social:



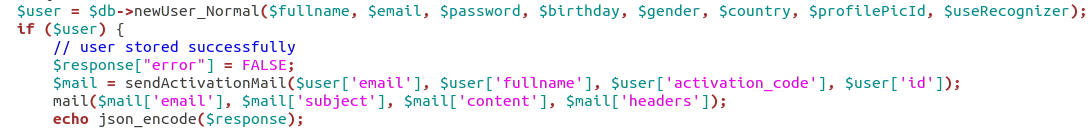
Utilizatorul va vedea mereu butonul conectează-te folosind facebook, pentru el este aceeași acțiune, diferența făcându-se la nivel de server, de aceea clientul va trimite de fiecare dată la conectare toate datele obținute de la API-ul facebook, iar dacă acesta este deja în tabelul de utilizatori nu va fi inserat din nou, altfel va fi adăugat și apoi se vor returna datele lui.

În cazul în care ceva merge greșit, parolă greșită sau email greșit, serverul va seta un flag de eroare.

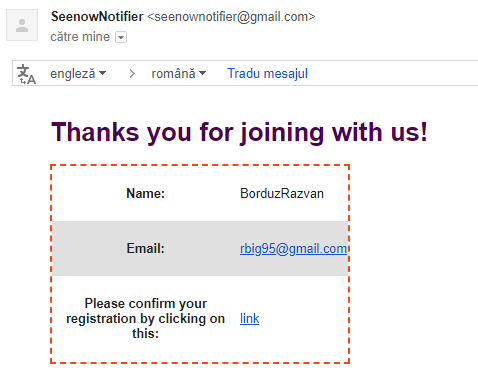
1. **Register.php**

În momentul înregistrării unui utilizator, se va apela metoda definită anterior numai dacă sunt oferite toate informațiile necesare pentru a se finaliza o înregistrare. Prima dată se va verifica dacă adresa de email există deja în baza de date, dacă nu există se va insera noul utilizator, apoi se va trimite un email de confirmare unde va fi un link către un alt script php pentru activarea contului ( **ștergerea codului de activare setat** ).





Mailul se va trimite folosind metoda mail din php și in plus acesta este configurat să folosească **smtp**, mai exact mailul de pe care se trimite este oferit de către google și în cazul înregistrării cu success mailul va arăta sub forma:



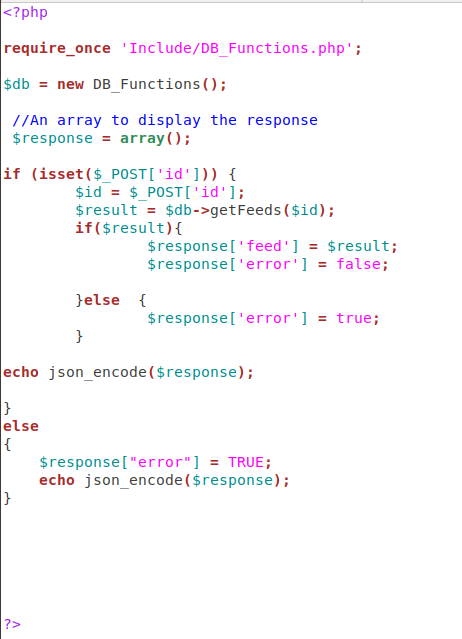
Funcția sendActivationMail va oferi parametrii cu care va fi chemată ulterior funcția php **mail** care va trimite efectiv mailul.



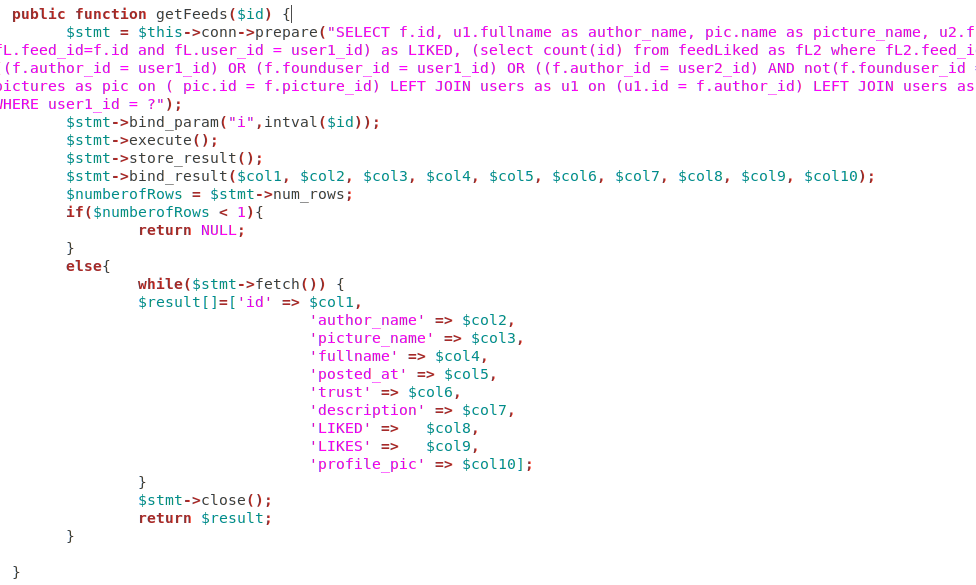
1. **Feed.php**

Are ca rol culegerea de informații pentru feed-ul unui utilizator, acestea fiind oferite sub formă de JSON pentru client.

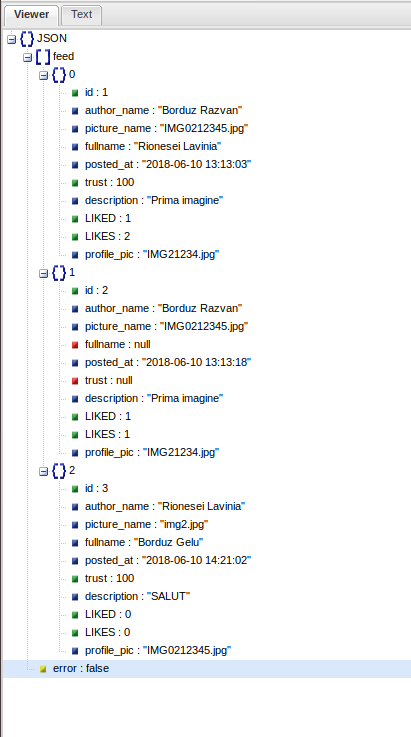
Intrarea pentru acest script php fiind id-ul utilizatorului pentru care se dorește obținerea feed-ului.



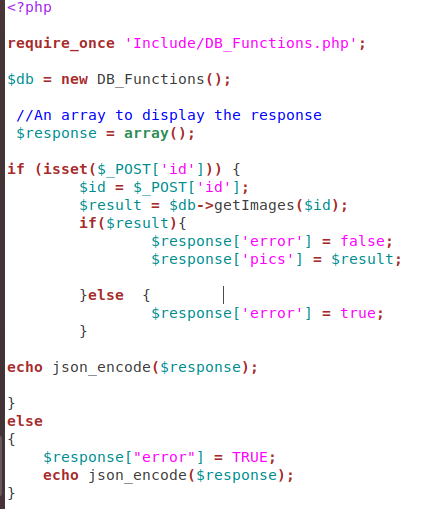
Iar funcția getFeeds va avea implementarea:

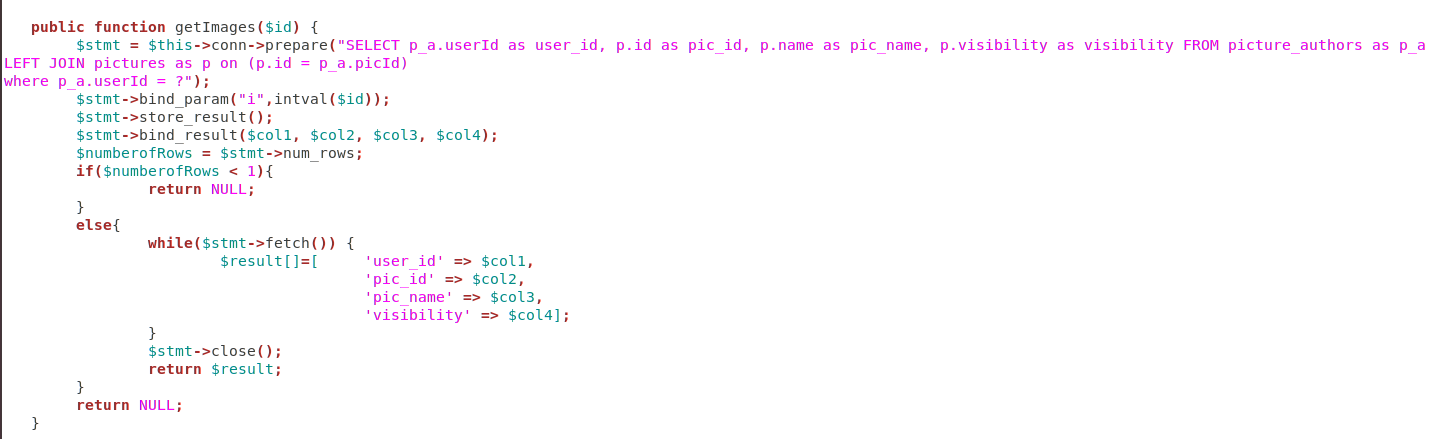


Se pregătește comanda care urmează a fi executată asupra bazei de date. Dacă memorează rezultatele în variabilele $coli , I = 1:10, dacă a fost cel puțin un rezultat se construiește array-ul result care urmează apoi a fi returnat scriptului feeds.php care la rândul lui îl va trimite ca și răspuns clientului.



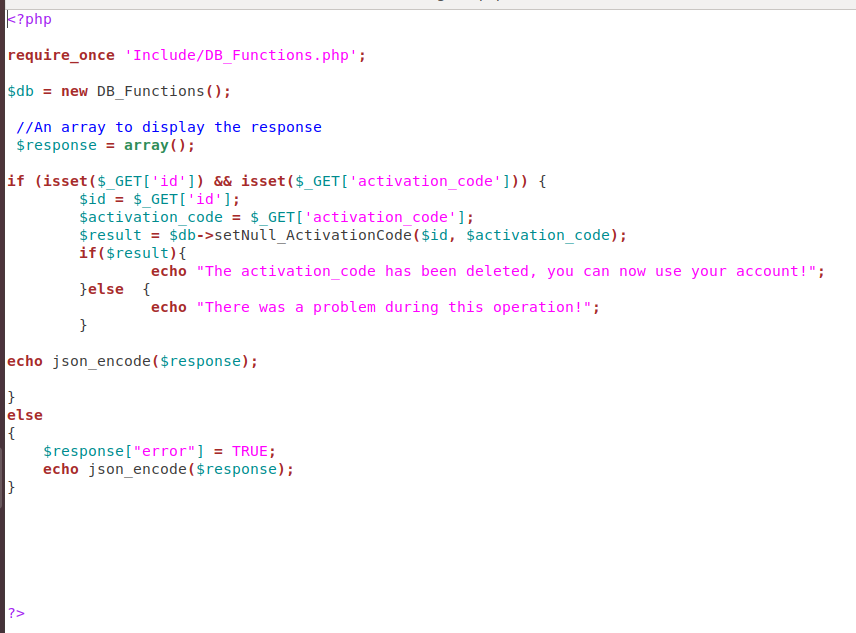
1. **Gallery.php** se va define similar ca și fișierul feed.php, cu diferența că se va apela getImages($id).

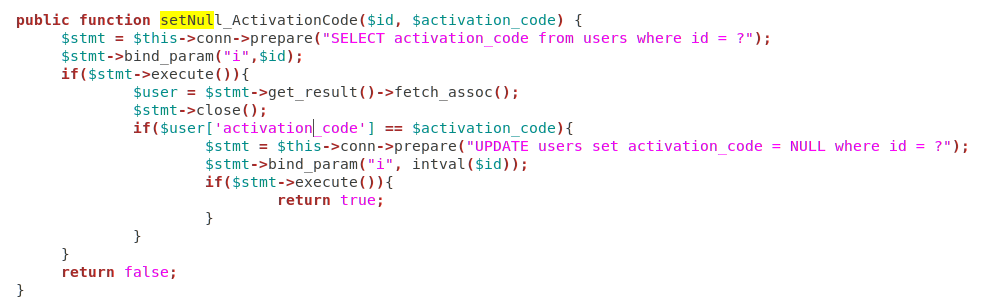




1. **Activation.php**

Pentru un anumit id și activation\_code oferit ca și intrare pentru script, șterge atributul activation\_code, dacă activation\_code-ul curent este egal cu cel oferit ca parametru pentru utilizatorul cu id-ul oferit ca parametru, de această dată informațiile fiind oferite prin metoda GET, deoarece apelul se va face direct din link-ul din email, de forma : <http://seenow.work/activation.php?id=314231&code=42314314.P>



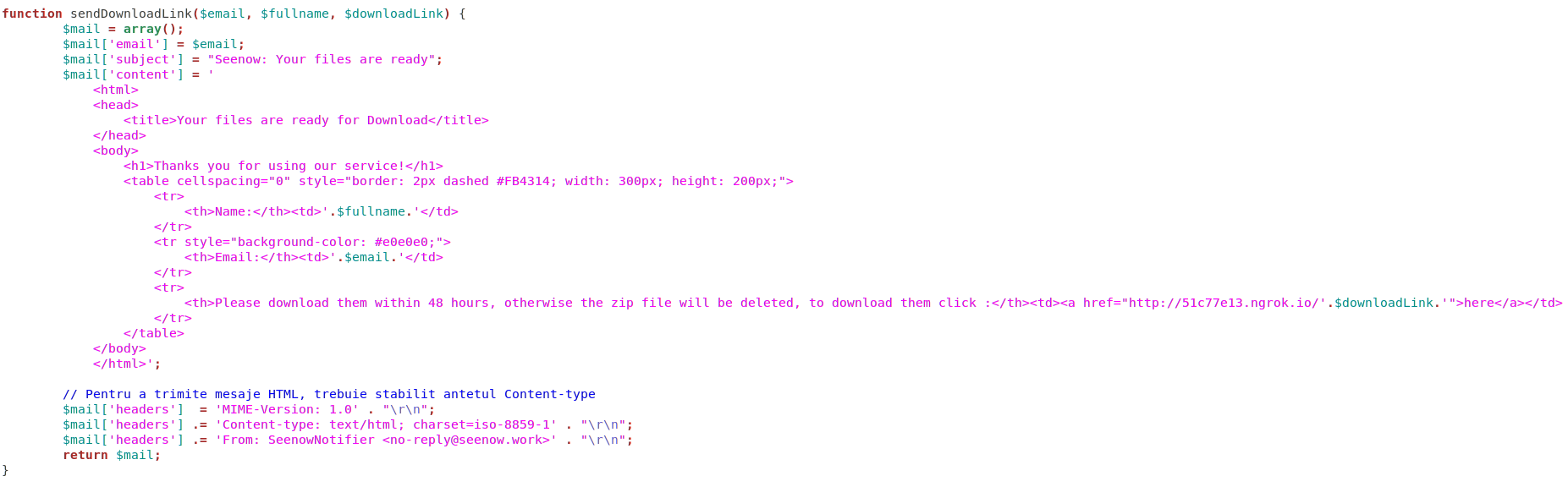


1. **Download.php**

Pentru un anumit id se va verifica existența lui în tabela utilizatori, iar apoi dacă acesta există se va creea o arhivă care va fi disponibilă pe server timp de 48 ore. După creearea arhivei utilizatorul va primi un email de confirmare cu privire la link-ul de descărcare al arhivei.



Iar funcția sendDownloadLink va genera informațiile care urmează a fi trimise utilizatorului pe mail folosind tag-uri HTML pentru a arăta mai frumos.



Iar folosind comanda:

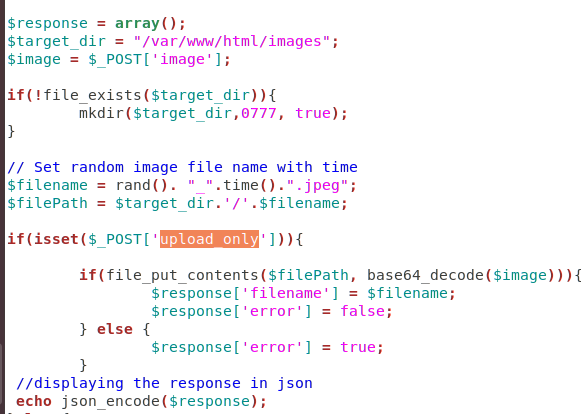
**@daily find /var/www/html/personIdentifier/Images/downloads/ -type f -mtime +2 -delete**

Se setează un thread care zilnic va șterge fișierele din mai vechi de 48h (2 zile).

1. **Upload.php**

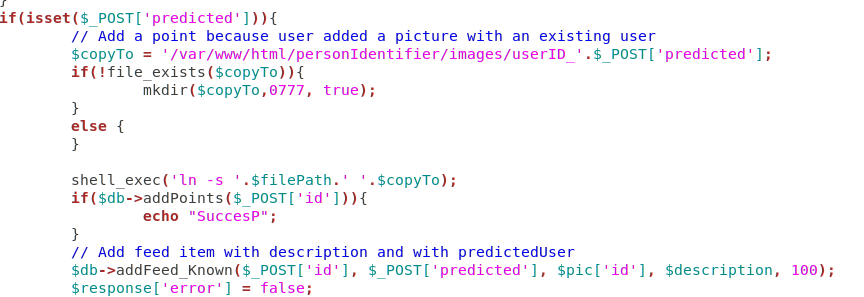
În cazul încărcării de imagini se descrie următoarea ordine a acțiunilor:

* Se încarcă numai imaginea fără a fi setat upload\_only și se va returna sub format json numele fișierului creeat în directorul images

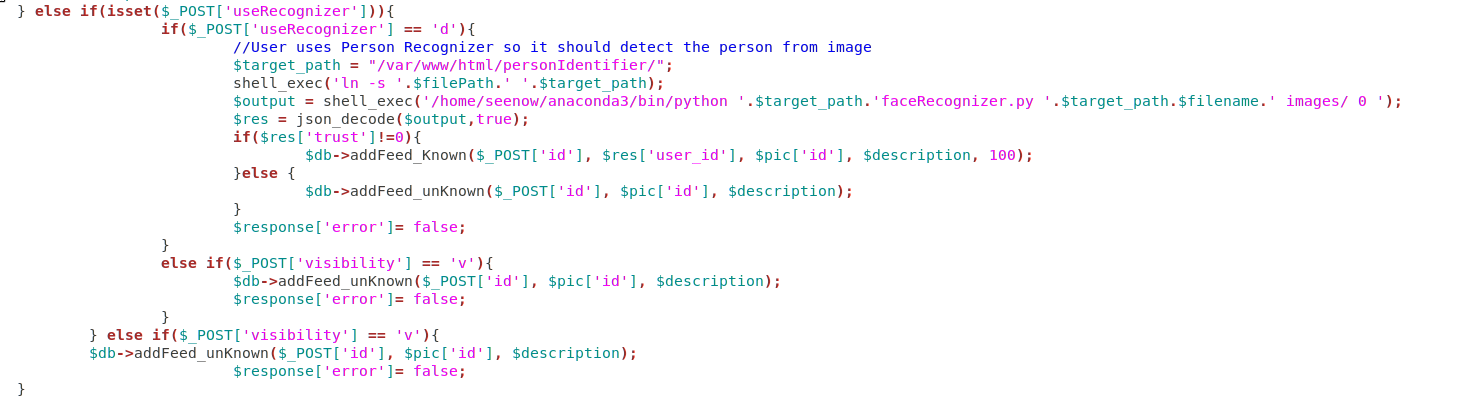


* Se apelează scriptul php upload.php oferind informați despre utilizator, vizibilitate, predicted\_user, descriere și numele fișierului, useRecognizer, iar în funcție de ele se vor introduce datele în tabele și la nevoie se va lansa scriptul python pentru recunoașterea utilizatorului din imagine.

Inserția în cazul în care utilizatorul este cunoscut deja.



Inserția în cazul în care utilizatorul nu este cunoscut, se va face după executarea scriptului python și evaluarea răspunsului oferit de către scriptul python.



1. **Actions.php** – acest fișier php are ca scop implementarea următoarelor servicii:

Schimbare parolă, schimbare adresă email, adăugare like element feed, ștergere element feed, recuperare parolă, schimbare poză profil. Pe baza cererii de la client va apela una din metodele care va prelucra baza de date și va oferi un răspuns tot sub formă de JSON.

### 3.2.3 Algoritmi de detecție și recunoaștere facială

Acest subcapitol este dedicat implementării python pentru detecția și recunoașterea facială folosind algoritmii: Fisherfaces și LPBH prezentați în capitolul **1.2.2.** Pe baza informațiilor cumulate în timp se va creea o bază de învățare care va fi actualizată periodic, încercându-se recunoașterea imaginilor care nu au fost recunoscute cu succes, valoarea normalizată a încrederii pe baza rezultatelor celor doi algoritmi nu depășește pragul de 70%.

Structura directoarelor va fi de forma:

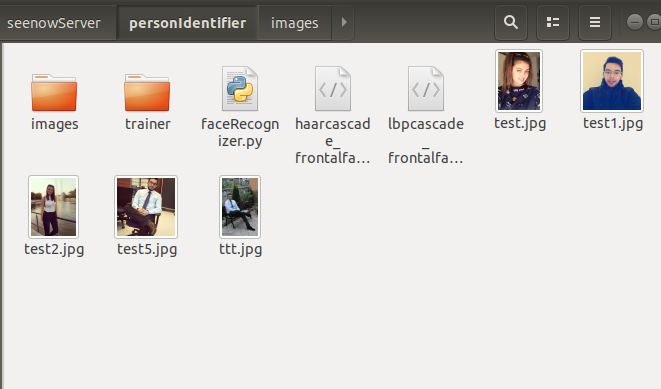


Figure 23 Strucuta director

Unde în directorul images se vor afla directoarele cu imagini specifice fiecărui utilizator, numele directorului fiind de forma: **userID\_X** – unde X este ID-ul unic al utilizatorului.

În directorul trainer se vor afla două fișiere trainer\_LBPH.yml și trainer\_FisherFaces.yml – fișere de antrenare care vor fi folosite în procesul de recunoaștere al persoanelor. Odată pe zi se va efectua actualizarea acestor fișere de antrenare care se creează în următorul fel:

* Intră în fiecare director și pentru fiecare imagine va atribui o etichetă egală cu ID-ul utilizatorului pentru prima față identificată în imagine, deoarece considerăm că în imagine se află numai o față. Informațiile obținute ulterior fiind salvate în fișierele anterior menționate pentru a fi folosite mai ușor, în momentul recunoașterii se va evalua numai fișierele de configurare și imaginea nouă, restul de imagini nefiind folosite, în acest fel se va reduce din timpul de executare.

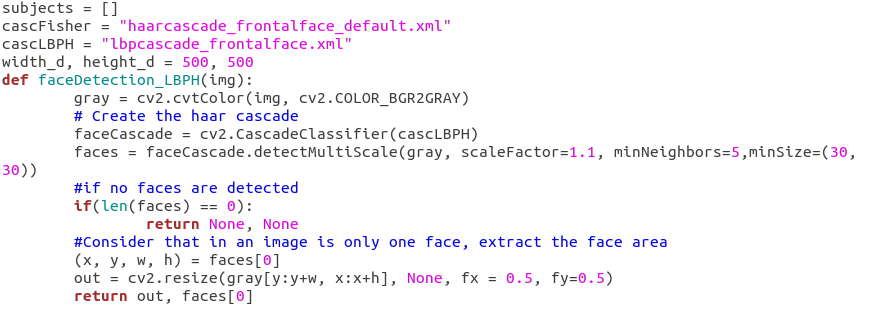
Scriptul python descris în fișierul FaceRecognizer.py necesită 3 argumente pentru a fi executat cu success: calea către imaginea nouă, calea către directorul cu setul de imagini, o valoare numerică 1 sau 0.

Dacă argumentul 3 are valoarea 0 atunci se va face numai recunoașterea imagini cu calea oferită în argumentul 1 pe baza cunoștințelor dobândite prin antrenare. (fișerele trainer).

Dacă argumentul are valoarea 1 atunci se va face numai procesul de antrenare a bazei de cunoștințe folosind numai calea către directorul cu setul de imagini obținând la final două fișere noi trainer\_LBPH.yml și trainer\_FisherFaces.yml

Un fir de execuție va executa periodic, odată la 24 de ore următoarea comandă pentru fiecare imagine din directorul images/unknown/

Python images/unknown/image.img trainer/ 0 executând procesul pentru imaginile în care nu a fost recunoscută nici o imagine.



cascFisher, cascLBPH – reprezintă căile către fișierele de cascadare folosite pentru detecția facială. În acest process se va considera dimensiunea maximă 500x500, efecuându-se un resize asupra imaginii feței în cazul algoritmului FisherFaces care nu acceptă dimensiuni mai mari.

Figure 24 Detecția folosind LBPH

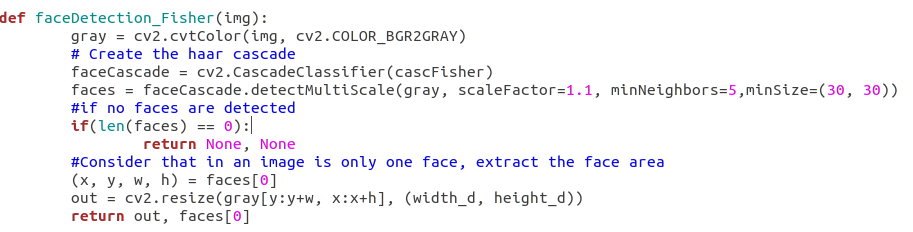


Figure 25 Detecția folosind FisherFaces

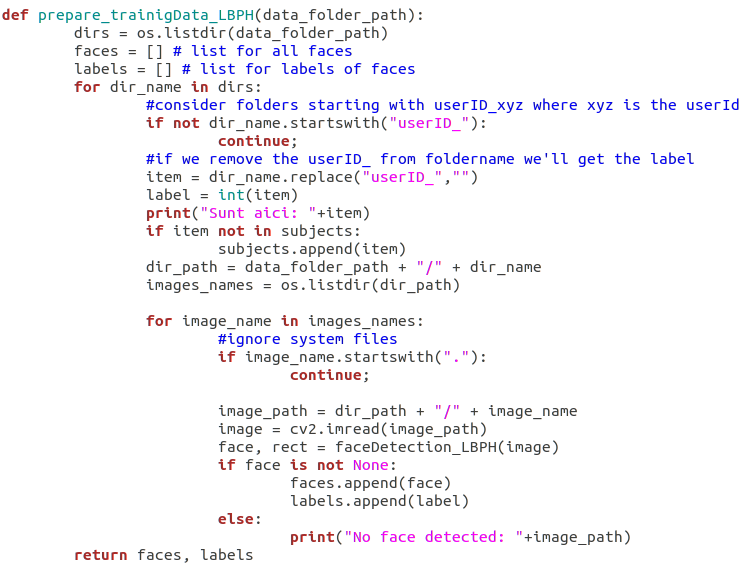


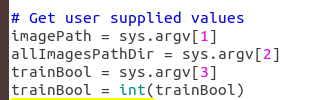
Figure 26 Funcția de antrenare pentru LBPH

Similar cu funcția de antrenare pentru LBPH a fost definită și funcția prepare\_trainingData\_Fisher(data\_folder\_path) însă care va folosi funcția faceDetection\_Fisher(image)



Similar cu predictLBPH se va defini și predictFisher(predict\_img) care va folosi face\_recognizer2.predict.

Citirea intrărilor pentru script-ul python:



Pe baza valorilor citite se vor efectua acțiunile prezentate anterior:

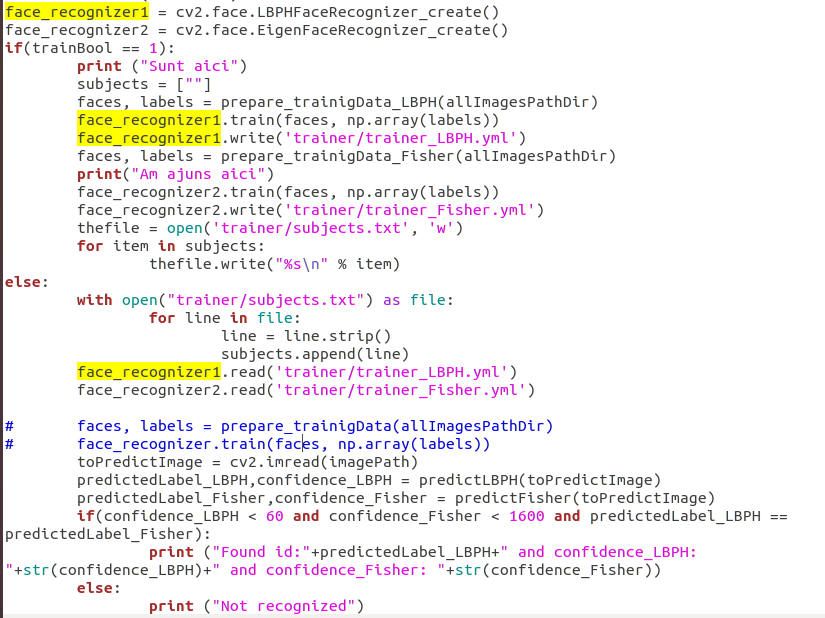


Figure 27 Apeluri faceRecognizer.py

Gradul de încredere rezultat al predicției pentru ambii algoritmi vor fi comparați cu anumite valori limită ( în figura 27 sunt date ca exemple 60 și 1600 urmând ca acestea să fie stabilite experimental ).

Pentru determinarea experimentală se va folosi un set de poze de 100 de imagini cu aceași persoană și se va observa valoarea care va fi returnată în urma unei analize corecte, respectiv valoarea în cazul în care fața altei persoane apare:

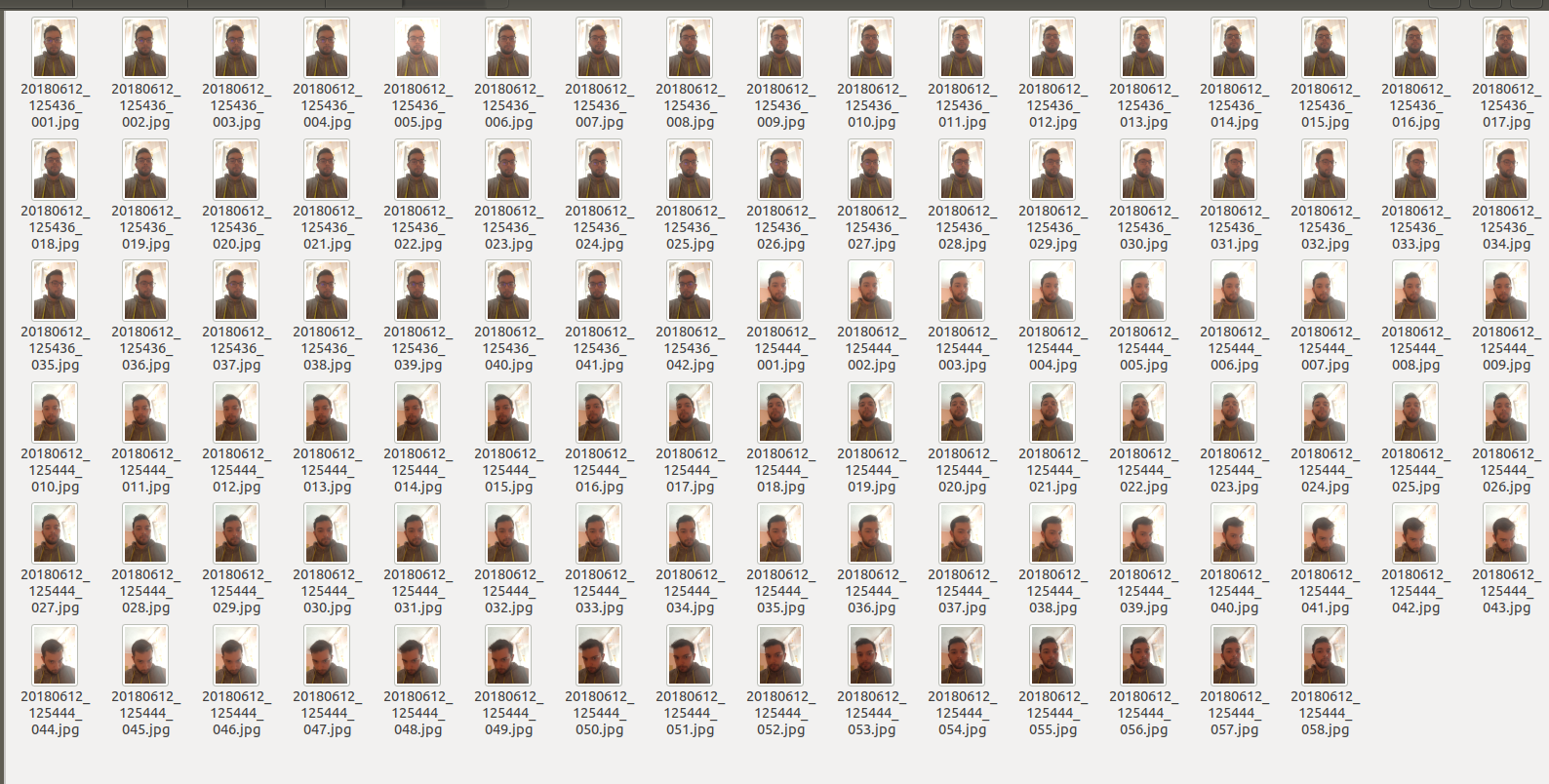
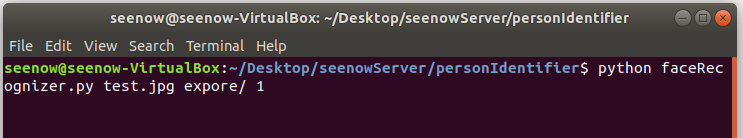


Figure 28 Setul de imagini experimentale

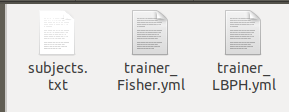
1. Executăm antrenarea folosind acest set de imagini.



Datorită luminii și a orientării feței algoritmul LBPH a eșuat în detectarea feței **în 80 de fețe.**

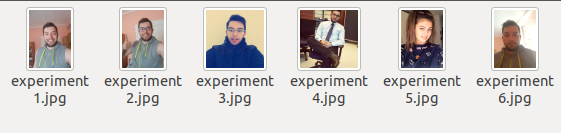
Datorită luminii și a orientării feței algoritmul FisherFaces a eșuat în detectarea feței **în 52 de fețe.**

În acest moment avem următoarele baze de cunoștințe:



În subjects.txt fiind id-urile utilizatorilor pentru care s-au verificat directoarele, în cazul nostru 1.

Acum vom testa baza noastră de învățare cu următoarele imagini:





|  |  |  |
| --- | --- | --- |
| Nume Imagine | Rezultat LBPH | Rezultat FisherFaces |
| Experiment1.jpeg | **44** | 21859 |
| Experiment2.jpeg | **41** | 19652 |
| Experiment3.jpg | 77 | 25732 |
| Experiment4.jpg | 153 | 17500 |
| Experiment5.jpg | 106 | 32509 |
| Experiment6.jpg | **0** | **0** |

Pe baza evaluărilor prezentate voi considera mai departe în proiect următoarele valori limită:

**LBPH** – 0 – 100% corect – peste 80 – incorect – funcția de corectitudine fiind fLBPH(x)=

**FisherFaces – 0 –** 100% corect – peste 20000 este incorect

FFisherFaces(x)=

Implementarea devenind astfel:



Dacă una dintre valori este pozitivă, se evaluează dacă ambele pointează către același id, dacă da se combină rezultatele: (fLBPH(x) + fFisherFaces(x)) / 2 , iar dacă rezultatul este peste 10 se va permite copierea, altfel se va considera corectă evaluarea cu coeficientul mai mare, fișierul de intrare fiind apoi mutat în folderul corespunzătr. Images/userID\_X sau images/unknown după caz.

Rezultatul scriptului python va fi oferit împachetat în format JSON scriptului php upload.php care apoi va insera informația în tabelul feeds apelând fie metoda add\_Known sau add\_unKnown.

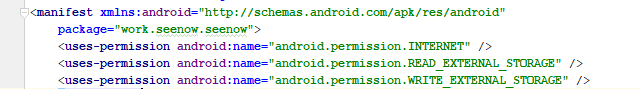
Apelul scriptului de detecție și recunoaștere facială devenind sub forma:

**Python faceRecognizer.py imagePath imageSet actionInt authorId description visibility**

### 3.3.4 Interfață utilizator

Interfața utilizator va fi descrisă de o aplicație Android. Pe parcursul acestui capitol

Pentru buna funcționare a aplicației sunt necesare următoarele permisiuni configurate în AndroidManifest.xml



Pe baza specificațiilor descrise la începutul capitolului 3 se vor distinge următoarele activități: **MainActivity, LoginActivity, RegisterActivity.** Pe lângă acestea vor fi descrise 3 4 fragmente: **FeedFragment**, **ProfileFragment**, **CameraFragment**, **SettingsFragment**.

Iar pentru implementarea funcțională în interiorul pachetului Utils se definesc mai multe clase support precum: AppConfig, User, FeedItem, FeedImageView, FeedListAdapter, SessionManager, SQLiteHandler, etc.

Toate asset-urile folosite în aplicație se vor găsi în directorul res: **drawables, strings, values, layout, menu.**

Aspectul activitățiilor și a elementelor care urmează a fi inserate în activitate vor fi descrise prin fișere xml care se vor afla în directorul layout.

Luând toate aceste aspecte în considerare structura proiectului Android Studio va arăta sub forma vezi figura 29

Toate librăriile utilizate în implementare se vor importa în fișierul de configurare: build.gradle (vezi figura 30), iar pe baza specificațiilor librăriilor utilizate am stabilit ca sdk-ul minim necesar pentru a utiliza aplicația să fie: 22, echivalentul versiunii de android 5.1 (vezi figura 31 )

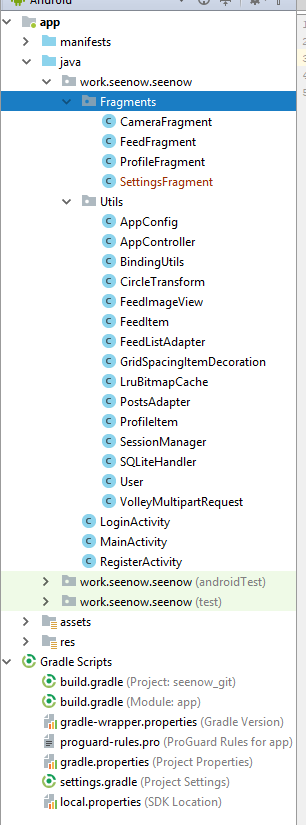
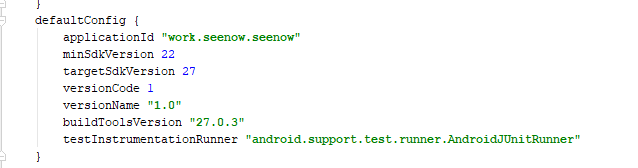


Figure 291 Versiune Android necesară

Figure 30 Librării utilizate în aplicația android

Figure 29 Structura proiect Android Studio

1. **MainActivity**

În această activitate se va decide conținutul ce urmează să fie afișat utilizatorului și se diferențiază două cazuri: Când utilizatorul este autentificat, respectiv când nu este, informația fiind salvată folosind clasa **SessionManager** care salvează în contextul aplicației folosind SharedPreferences și un editor, oferind metodele setLogin pentru modificarea stării, respectiv isLoggedIn pentru verificarea stării.

**public class** SessionManager {  
 *// LogCat tag* **private static** String *TAG* = SessionManager.**class**.getSimpleName();  
  
 *// Shared Preferences* SharedPreferences **pref**;  
 Editor **editor**;  
 Context **\_context**;  
 *// Shared pref mode* **int PRIVATE\_MODE** = 0;  
 *// Shared preferences file name* **private static final** String ***PREF\_NAME*** = **"SeenowLogin"**;  
 **private static final** String ***KEY\_IS\_LOGGEDIN*** = **"isLoggedIn"**;  
  
 **public** SessionManager(Context context) {  
 **this**.**\_context** = context;  
 **pref** = **\_context**.getSharedPreferences(***PREF\_NAME***, **PRIVATE\_MODE**);  
 **editor** = **pref**.edit();  
 }  
  
 **public void** setLogin(**boolean** isLoggedIn) {  
  
 **editor**.putBoolean(***KEY\_IS\_LOGGEDIN***, isLoggedIn);  
  
 *// commit changes* **editor**.commit();  
  
 Log.*d*(*TAG*, **"User login session modified!"**);  
 }  
  
 **public boolean** isLoggedIn(){  
 **return pref**.getBoolean(***KEY\_IS\_LOGGEDIN***, **false**);  
 }  
}

Informațiile despre utilizator vor fi salvate într-o bază de date locale SQLite, care va stoca informații despre utilizator în tabelul user din baza de date seenow\_db, informații precum: id, fullname, email, birthday, profilePicture, etc. Implementarea pentru baza de date locală SQLite se va face în clasa SQLiteHandler în care pentru fiecare modificare asupra structurii tabelei se va incrementa versiunea bazei de date:   
*// Database Version***private static final int *DATABASE\_VERSION*** = 11;

Când aplicația va observa o versiunea diferită a bazei de date față de cea care este în memorie va apela metoda onUpgrade care va șterge tabelul users și îl va creea din nou.

Clasa SQLiteHandler oferă funcțiile de inserție, extracție, actualizare și ștergere utilizatori.

**public class** SQLiteHandler **extends** SQLiteOpenHelper {  
  
 **private static final** String ***TAG*** = SQLiteHandler.**class**.getSimpleName();  
 *// All Static variables  
 // Database Version* **private static final int *DATABASE\_VERSION*** = 11;  
  
 *// Database Name* **private static final** String ***DATABASE\_NAME*** = **"seenow\_db"**;  
  
 *// Login table name* **private static final** String ***TABLE\_USER*** = **"user"**;  
  
 *// Login Table Columns names* **private static final** String ***KEY\_ID*** = **"id"**;  
 **private static final** String ***KEY\_NAME*** = **"fullname"**;  
 **private static final** String ***KEY\_EMAIL*** = **"email"**;  
 **private static final** String ***KEY\_PROFILE\_PIC*** = **"profilePicture"**;  
 **private static final** String ***KEY\_COUNTRY*** = **"country"**;  
 **private static final** String ***KEY\_POINTS*** = **"points"**;  
 **private static final** String ***KEY\_USE\_REC*** = **"use\_Recognizer"**;  
 **private static final** String ***KEY\_GENDER*** = **"gender"**;  
 **private static final** String ***KEY\_BIRTHDAY*** = **"birthday"**;  
 **private static final** String ***KEY\_NR\_PICTURES*** = **"nr\_pictures"**;  
 **private static final** String ***KEY\_NR\_friends*** = **"nr\_friends"**;  
 **private static final** String ***KEY\_NR\_foundIn*** = **"nr\_foundIn"**;  
 **private static final** String ***KEY\_CREATED\_AT*** = **"created\_at"**;  
 **private static final** String ***KEY\_ABOUT*** = **"about"**;  
  
 **public** SQLiteHandler(Context context) {  
 **super**(context, ***DATABASE\_NAME***, **null**, ***DATABASE\_VERSION***);  
 }  
  
 *// Creating Tables* @Override  
 **public void** onCreate(SQLiteDatabase db) {  
 String CREATE\_LOGIN\_TABLE = **"CREATE TABLE "** + ***TABLE\_USER*** + **"("** + ***KEY\_ID*** + **" INTEGER PRIMARY KEY,"** + ***KEY\_NAME*** + **" TEXT,"** + ***KEY\_EMAIL*** + **" TEXT UNIQUE,"** + ***KEY\_PROFILE\_PIC*** + **" TEXT,"** + ***KEY\_BIRTHDAY*** + **" TEXT,"** + ***KEY\_COUNTRY*** + **" TEXT, "** + ***KEY\_GENDER*** + **" TEXT,"** + ***KEY\_POINTS***+ **" INTEGER, "** + ***KEY\_USE\_REC*** + **" TEXT,"** + ***KEY\_NR\_PICTURES*** + **" INTEGER, "** + ***KEY\_NR\_foundIn*** +**" INTEGER, "**+ ***KEY\_NR\_friends*** + **" INTEGER, "** + ***KEY\_CREATED\_AT*** +**" TEXT, "** + ***KEY\_ABOUT*** + **" TEXT)"**;  
 db.execSQL(CREATE\_LOGIN\_TABLE);  
  
 Log.*d*(***TAG***, **"Database tables created si am string-ul "**+CREATE\_LOGIN\_TABLE);  
 }  
  
 *// Upgrading database* @Override  
 **public void** onUpgrade(SQLiteDatabase db, **int** oldVersion, **int** newVersion) {  
 *// Drop older table if existed* db.execSQL(**"DROP TABLE IF EXISTS "** + ***TABLE\_USER***);  
  
 *// Create tables again* onCreate(db);  
 }

**public void** addUser(User u) {  
 SQLiteDatabase db = **this**.getWritableDatabase();  
  
 ContentValues values = **new** ContentValues();  
 values.put(***KEY\_NAME***, u.getName());  
 values.put(***KEY\_EMAIL***, u.getEmail());  
 values.put(***KEY\_BIRTHDAY***, u.getBirthday());  
 values.put(***KEY\_PROFILE\_PIC***, u.getProfileImage());  
 values.put(***KEY\_COUNTRY***, u.getCountry());  
 values.put(***KEY\_POINTS***, u.getnumberofPoints().get());  
 values.put(***KEY\_USE\_REC***, u.getUseRecognizer());  
 values.put(***KEY\_GENDER***, u.getGender());  
 values.put(***KEY\_ID***, u.getId());  
 values.put(***KEY\_NR\_foundIn***, u.**numberofAppereances**.get());  
 values.put(***KEY\_NR\_PICTURES***, u.**numberofPhotosTaken**.get());  
 values.put(***KEY\_NR\_friends***, u.**numberofFriends**.get());  
 values.put(***KEY\_ABOUT***, u.getAbout());  
  
 *// Inserting Row* **long** id = db.insert(***TABLE\_USER***, **null**, values);  
 db.close(); *// Closing database connection* Log.*d*(***TAG***, **"New user inserted into sqlite: "** + id);  
}

Funcția de inserție – **addUser** care primește ca parametru un obiect de tipul User, va insera în tabel informațiile pe care le obține de la acel utilizator prin intermediul ContentValues în care se vor insera perechi: (cheie,valoare).

**public** User getUserDetails() {  
 String selectQuery = **"SELECT \* FROM "** + ***TABLE\_USER***;  
  
 SQLiteDatabase db = **this**.getReadableDatabase();  
 Cursor cursor = db.rawQuery(selectQuery, **null**);  
 *// Move to first row* cursor.moveToFirst();  
 **if** (cursor.getCount() > 0) {  
  
  
 User user = **new** User(cursor.getInt(0), cursor.getString(1),  
 cursor.getString(2),(cursor.getString(3)).substring(AppConfig.*URL\_SERVER*.length() -1),  
 cursor.getString(4), cursor.getString(5),  
 cursor.getString(6), cursor.getInt(7),  
 cursor.getString(8),cursor.getString(12), cursor.getString(13));  
 user.**numberofAppereances**.set(cursor.getLong(10));  
 user.**numberofPhotosTaken**.set(cursor.getLong(9));  
 user.**numberofFriends**.set(cursor.getLong(11));  
 **return** user;  
 }**else** {  
 **return null**;  
 }  
}

Funcția de extracție – **getUserDetails** care va returna un obiect de tipul User cu informațiile din primul rând obținut în urma comenzii de select \* from users, altfel va returna null.

**public void** updateUsers(User u){  
 SQLiteDatabase db = **this**.getWritableDatabase();  
  
 ContentValues values = **new** ContentValues();  
 values.put(***KEY\_NAME***, u.getName());  
 values.put(***KEY\_BIRTHDAY***, u.getBirthday());  
 values.put(***KEY\_PROFILE\_PIC***, u.getProfileImage());  
 values.put(***KEY\_COUNTRY***, u.getCountry());  
 values.put(***KEY\_POINTS***, u.getnumberofPoints().get());  
 values.put(***KEY\_USE\_REC***, u.getUseRecognizer());  
 values.put(***KEY\_GENDER***, u.getGender());  
 values.put(***KEY\_NR\_foundIn***, u.**numberofAppereances**.get());  
 values.put(***KEY\_NR\_PICTURES***, u.**numberofPhotosTaken**.get());  
 values.put(***KEY\_NR\_friends***, u.**numberofFriends**.get());  
 values.put(***KEY\_ABOUT***, u.getAbout());  
 String where=***KEY\_ID***+**"=?"**;  
 String[] args = **new** String[] {String.*valueOf*(u.getId())};  
 db.update(***TABLE\_USER***,values,where,args);  
 *// Inserting Row* **long** id = db.insert(***TABLE\_USER***, **null**, values);  
 db.close(); *// Closing database connection* Log.*d*(***TAG***, **"user updated into sqlite: "** + id);  
 }

Funcția de actualizare – **updateUsers** care primește ca parametru un obiect de tipul User și va actualiza informațiile înregistrării din tabelul users care are id-ul egal cu cel obținut de la obiectul de tip Users.

**public void** deleteUsers() {  
 SQLiteDatabase db = **this**.getWritableDatabase();  
 *// Delete All Rows* db.delete(***TABLE\_USER***, **null**, **null**);  
 db.close();  
  
 Log.*d*(***TAG***, **"Deleted all user info from sqlite"**);  
}

Clasa User va extinde clasa BaseObservable, deoarece atributele unui utilizator vor modifica elementele interfeței utilizator folosind databinding. Data Binding reprezintă o librărie suport care permite legarea direct a componentelor UI la elemente sursă, precum variabile, fără a face ulterior actualizarea acestora.

Pentru a folosi databinding am modificat build.grade-ul aplicației și am adăugat linia:

dataBinding {  
 enabled = **true**}

**public class** User **extends** BaseObservable **implements** Serializable {  
 **private int id**;  
 **private** String **name**;  
 **private** String **email**;  
 **private** String **profileImage**;  
 **private** String **birthday**;  
 **private** String **country**;  
 **private** String **gender**;  
 **private** String **useRecognizer**;  
 **private** String **created\_at**;  
 **private** String **about**;

Constructorul obiectului arată sub forma:

**public** User(**int** id, String name, String email, String profileImage, String birthday, String country, String gender, **int** points, String userRecognizer, String created\_at, String about) {  
 **this**.**id** = id;  
 **this**.**name** = name;  
 **this**.**email** = email;  
 **this**.**profileImage** = AppConfig.*URL\_SERVER*+profileImage;  
 Log.*d*(**"UserClass"**,**"Am profileLink: "**+**this**.**profileImage**);  
 **this**.**birthday** = birthday;  
 **this**.**country** = country;  
 **this**.**gender** = gender;  
 **numberofPoints**.set(points);  
 **this**.**useRecognizer** = userRecognizer;  
 **this**.**created\_at** = created\_at;  
 **this**.**about** = about;  
}

Am concatenat profileImage care reprezintă numele fișierului cu **AppConfig.URL\_SERVER** care reprezintă: calea către directorul unde se află imaginile pe server. Atributele numberofPhotosTaken, numberofAppereances, numberofFriends, numberofPoints vor actualiza UI-ul fără a exista vreo metodă de get definită manual.

**public** ObservableField<Long> **numberofPhotosTaken** = **new** ObservableField<>();  
**public** ObservableField<Long> **numberofAppereances** = **new** ObservableField<>();  
**public** ObservableField<Long> **numberofFriends** = **new** ObservableField<>();  
**public** ObservableField<Integer> **numberofPoints** = **new** ObservableField<>();

Am definit pentru fiecare atribut al obiectului de tip User o metodă de get, respective set a valorii atributului.

Atributele care vor modifica UI-ul vor avea adnotația : @Bindable în fața metodei de get.

@Bindable  
**public** String getAbout() { **return this**.**about**;}

Setarea elementului UI de tip ImageView în cazul utilizatorului care va reprezenta imaginea lui de profil se va face folosind libăria Glide. Pentru a putea folosi librăria Glide am adăugat linia următoare în build.gradle:   
implementation **'com.github.bumptech.glide:glide:3.7.0'**

Iar metoda de loadImage care va afecta în mod direct UI-ul va arăta sub forma:

@BindingAdapter({**"profileImage"**})  
**public static void** loadImage(ImageView view, String imageUrl) {  
 Glide.*with*(view.getContext())  
 .load(imageUrl)  
 .crossFade()  
 .thumbnail(0.5f)  
 .bitmapTransform(**new** CircleTransform(view.getContext()))  
 .into(view);  
}

.crossFade() reprezintă folosirea unei animații în momentul schimbării imaginii pentru a avea un aspect mai plăcut.

.thumbnail(0.5f) reprezintă faptul că imaginea originală va fi afișată la 50% din dimensiunea sa.

Clasa CirleTransform având ca scop definirea metodei de transformare a imaginii sub forma



Figure 31 Imagine preluată din aplicația See Now

Clasa AppConfig va conține toate adresele către scripturile php care vor rula pe server:

**public class** AppConfig {  
 *// Server user login url* **public static** String *URL\_LOGIN* = **"http://e87978a3.ngrok.io/login.php"**;  
  
 *// Server user register url* **public static** String *URL\_REGISTER* = **"http://e87978a3.ngrok.io/register.php"**;  
  
 *// Server feed url* **public static** String *URL\_FEED* = **"http://e87978a3.ngrok.io/feed.php"**;  
  
 *// Server Image url* **public static** String *URL\_SERVER* = **"http://e87978a3.ngrok.io/images/"**;  
  
 *// Server Actions url* **public static** String *URL\_ACTIONS* =**"http://e87978a3.ngrok.io/actions.php"**;  
  
 *// Server Gallery Url* **public static** String *URL\_GALLERY* =**"http:///e87978a3.ngrok.io/gallery.php"**;  
  
 *// Server Upload Url* **public static** String *URL\_UPLOAD*= **"http:///e87978a3.ngrok.io/upload.php"**;  
 *// Place for SeeNow Picture* **public static final** String ***IMAGE\_DIRECTORY\_NAME*** = **"SeenowPictures"**;  
}

Unde “http://e87978a3.ngrok.io reprezintă domeniul oferit de către aplicația ngrok serverului linux pentru a fi accesat mai ușor.

Acum că am definit clasele suport care urmează a fi folosite în implementarea clasei MainActivity urmează să definesc layoutul acesteia. Acesta va fi alcătuit dintr-un **DrawerLayout**, în interiorul lui fiind un **NavigationView**.

<**android.support.v4.widget.DrawerLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:app="http://schemas.android.com/apk/res-auto"  
 xmlns:tools="http://schemas.android.com/tools"  
 android:id="@+id/drawer\_layout"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:fitsSystemWindows="true"  
 tools:openDrawer="start"**>  
  
 <**include  
 layout="@layout/app\_bar\_main"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"** />  
  
 <**android.support.design.widget.NavigationView  
 android:id="@+id/nav\_view"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="match\_parent"  
 android:layout\_gravity="start"  
 android:fitsSystemWindows="true"  
 app:headerLayout="@layout/nav\_header\_main"  
 app:menu="@menu/drawer\_view"** />  
  
</**android.support.v4.widget.DrawerLayout**>

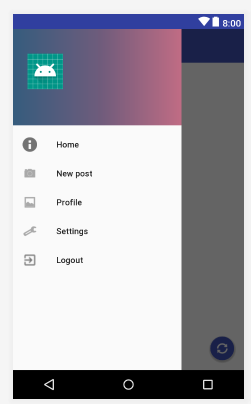


Figure 32 NavigationView SeeNow

Include layout=”@layout/app\_bar\_main” adaugă elementele descrise în acel layout în layoutul main\_activity, unde FrameLayout reprezintă layoutul în care va fi încărcat ulterior layoutul fragmentelor. Conținutul app\_bar\_main fiind:

<**android.support.design.widget.CoordinatorLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:app="http://schemas.android.com/apk/res-auto"  
 xmlns:tools="http://schemas.android.com/tools"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:fitsSystemWindows="true"  
 tools:context=".MainActivity"**>  
  
 <**android.support.design.widget.AppBarLayout  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:theme="@style/AppTheme.AppBarOverlay"**>  
  
 <**android.support.v7.widget.Toolbar  
 android:id="@+id/toolbar"  
 android:layout\_width="match\_parent"  
 android:layout\_height="?attr/actionBarSize"  
 android:background="?attr/colorPrimary"  
 app:popupTheme="@style/AppTheme.PopupOverlay"** />  
  
 </**android.support.design.widget.AppBarLayout**>  
  
 <**FrameLayout  
 android:id="@+id/frame"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 app:layout\_behavior="@string/appbar\_scrolling\_view\_behavior"**></**FrameLayout**>  
  
 <**android.support.design.widget.FloatingActionButton  
 android:id="@+id/fab"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:layout\_gravity="bottom|end"  
 android:layout\_margin="@dimen/fab\_margin"  
 app:backgroundTint="@color/colorPrimary"  
 app:srcCompat="@android:drawable/ic\_popup\_sync"** />  
  
</**android.support.design.widget.CoordinatorLayout**>

Atributul **headerLayout** (**app:headerLayout="@layout/nav\_header\_main"**) reprezintă layoutul din zona superioară a NavigationView-ului care va conține două ImageView-uri. Un ImageView va conține imaginea de fundal a zonei superioare, iar celălalt ImageView va fi folosit pentru setarea imaginii de profil a utilizatorului. Pe lângă acestea layout-ul nav\_header\_main va conține și un TextView în care va fi afișat numele utilizatorului.

<**RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:app="http://schemas.android.com/apk/res-auto"  
 android:id="@+id/view\_container"  
 android:layout\_width="match\_parent"  
 android:layout\_height="@dimen/nav\_header\_height"  
 android:gravity="bottom"  
 android:orientation="vertical"  
 android:theme="@style/ThemeOverlay.AppCompat.Dark"**>  
  
 <**ImageView  
 android:id="@+id/img\_header\_bg"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:scaleType="fitXY"  
 android:background="@drawable/gradient"** />  
  
 <**LinearLayout  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:layout\_centerVertical="true"  
 android:orientation="vertical"  
 android:padding="@dimen/activity\_horizontal\_margin"**>  
  
 <**ImageView  
 android:id="@+id/img\_profile"  
 android:layout\_width="@dimen/profile\_width"  
 android:layout\_height="@dimen/profile\_height"  
 android:paddingTop="@dimen/nav\_header\_vertical\_spacing"  
 app:srcCompat="@android:drawable/sym\_def\_app\_icon"** />  
  
 <**TextView  
 android:id="@+id/name"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:paddingTop="@dimen/nav\_header\_vertical\_spacing"  
 android:textAppearance="@style/TextAppearance.AppCompat.Body1"** />  
  
 </**LinearLayout**>  
</**RelativeLayout**>

Atributul **menu** al navigation view-ului reprezntă calea către xml-ul de configurație de tip menu în care se vor afla toate elementele navigation view-ului.

*<?***xml version="1.0" encoding="utf-8"***?>*<**menu xmlns:android="http://schemas.android.com/apk/res/android"**>  
  
<**group android:checkableBehavior="single"**>  
 <**item  
 android:id="@+id/nav\_Home"  
 android:icon="@android:drawable/ic\_dialog\_info"  
 android:title="Home"** />  
 <**item  
 android:id="@+id/nav\_NewPost"  
 android:icon="@android:drawable/ic\_menu\_camera"  
 android:title="@string/Newpost"** />  
 <**item  
 android:id="@+id/nav\_MyProfile"  
 android:icon="@android:drawable/ic\_menu\_gallery"  
 android:title="@string/profile"** />  
 <**item  
 android:id="@+id/nav\_Settings"  
 android:icon="@android:drawable/ic\_menu\_manage"  
 android:title="@string/settings"** />  
 <**item  
 android:id="@+id/nav\_logout"  
 android:icon="@drawable/exit"  
 android:title="@string/logout"** />  
</**group**>  
</**menu**>

Pentru fiecare acțiune din meniu în interiorul clasei MainActivity definesc un TAG cu care să identific elementul selectat din meniu: **private static final** String ***TAG\_HOME*** = **"home"**;  
**private static final** String ***TAG\_SETTINGS*** = **"settings"**;  
**private static final** String ***TAG\_PROFILE*** = **"profile"**;  
**private static final** String ***TAG\_NEW\_POST*** = **"new\_post"**;

TAG-ul elementului selectat va fi reținut în:

**public static** String *CURRENT\_TAG* = ***TAG\_HOME***;

**public class** MainActivity **extends** AppCompatActivity {  
 **private static final** String ***TAG*** = MainActivity.**class**.getSimpleName();  
  
 **private** NavigationView **navigationView**;  
 **public static** SQLiteHandler *db*;  
 **private** SessionManager **session**;  
 **private** DrawerLayout **drawer**;  
 **private** View **navHeader**;  
 **private** ImageView **imgNavHeaderBg**, **imgProfile**;  
 **private** TextView **txtName**;  
 **private** Toolbar **toolbar**;  
 **private** FloatingActionButton **fab**;  
 **private int target\_user**;  
  
 *// index to identify current nav menu item* **public static int** *navItemIndex* = 0;  
  
 **private static final** String ***TAG\_HOME*** = **"home"**;  
 **private static final** String ***TAG\_SETTINGS*** = **"settings"**;  
 **private static final** String ***TAG\_PROFILE*** = **"profile"**;  
 **private static final** String ***TAG\_NEW\_POST*** = **"new\_post"**;  
  
 **public static** String *CURRENT\_TAG* = ***TAG\_HOME***;  
 **private static** User *user*;  
 *// toolbar titles respected to selected nav menu item* **private** String[] **activityTitles**;  
  
 *// flag to load home fragment when user presses back key* **private boolean shouldLoadHomeFragOnBackPress** = **true**;  
 **private** Handler **mHandler**;

Metoda onCreate a clasei MainActivity este:

@Override  
**protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(R.layout.***activity\_main***);  
  
 *// SqLite database handler  
 db* = **new** SQLiteHandler(getApplicationContext());

// Extragem utilizatorul din tabela daca exista  
 *user*= *db*.getUserDetails();  
  
 *// session manager* **session** = **new** SessionManager(getApplicationContext());

// În cazul în care utilizatorul nu este autentificat se cheamă metoda logoutUser()  
 **if** (!**session**.isLoggedIn() || *user* == **null**) {  
 Log.*d*(***TAG***,**"SUNT AICI"**);  
 logoutUser();  
 } **else** {

// Utilizatorul este autentificat   
 Log.*d*(***TAG***, **"UserName: "**+*user*.getName()+**"id: "**+*user*.getId());  
  
 Log.*d*(***TAG***, **"is loggedin"**);  
  
 */\*\* User is logged in \*/* **toolbar** = (Toolbar) findViewById(R.id.***toolbar***);  
 setSupportActionBar(**toolbar**);  
  
 **mHandler** = **new** Handler();  
  
 **drawer** = (DrawerLayout) findViewById(R.id.***drawer\_layout***);  
 **navigationView** = (NavigationView) findViewById(R.id.***nav\_view***);  
 **fab** = (FloatingActionButton) findViewById(R.id.***fab***);  
  
 *// Navigation view header* **navHeader** = **navigationView**.getHeaderView(0);  
 **txtName** = (TextView) **navHeader**.findViewById(R.id.***name***);  
 **imgNavHeaderBg** = (ImageView) **navHeader**.findViewById(R.id.***img\_header\_bg***);  
 **imgProfile** = (ImageView) **navHeader**.findViewById(R.id.***img\_profile***);  
  
 // Incarcam titlurile pentru toolbar-ul Navigation View-ului.  
**activityTitles** = getResources().getStringArray(R.array.***nav\_item\_activity\_titles***);

// În cazul în care fab este apăsat se va efectua refresh asupra //fragmentului FeedFragments  
 **fab**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View view) {  
 Snackbar.*make*(view, **"Feed Refresh"**, Snackbar.***LENGTH\_LONG***)  
 .setAction(**"Action"**, **null**).show();  
 FeedFragment.*getFeeds*();  
 }  
 });  
  
  
 *// Încarc datele în zona superioară a navigation view-ului* loadNavHeader(*user*.getProfileImage());  
  
 *// Initializare navigation view* setUpNavigationView();

// În cazul în care nu s-a salvat nici o informație, se trece la fragmentul // reprezentat de TAG\_HOME: FeedFragment   
 **if** (savedInstanceState == **null**) {  
 *navItemIndex* = 0;  
 *CURRENT\_TAG* = ***TAG\_HOME***;

**// Se încarcă Fragmentul în Activitate**   
 loadHomeFragment();  
 }  
  
 }  
}

Metoda loadNavHeader va încărca informațiile utilizatorului în layoutul nav\_header\_main

**public void** loadNavHeader(String userImage) {  
 *// Setez textul care va fi afișat în TextView cu numele utilizatorului* **txtName**.setText(*user*.getName());  
  
 *// Încarc imaginea de fundal a zonei superioare* **imgNavHeaderBg**.setImageResource(R.drawable.***nav\_menu\_header\_bg***);  
  
 *// Încarc imaginea de profil care va fi afișată în zona superioară*

*// Databinding-ul fiind aplicat în cazul profilului.* Glide.*with*(**this**).load(userImage)  
 .crossFade()  
 .thumbnail(0.5f)  
 .bitmapTransform(**new** CircleTransform(**this**))  
 .diskCacheStrategy(DiskCacheStrategy.***ALL***)  
 .into(**imgProfile**);  
}

Metoda loadHomeFragment va încărca un anumit fragment pe baza elementului selectat din navigation view:

**private void** loadHomeFragment() {  
 *// selectează elementul correct din meniu* selectNavMenu();  
  
 *// setează titlul din toolbar* setToolbarTitle();  
  
 *// Dacă utilizatorul a selectat același element nu efectua nici o modificare* **if** (getSupportFragmentManager().findFragmentByTag(*CURRENT\_TAG*) != **null**) {  
 **drawer**.closeDrawers();  
  
 *// Afișează sau ascunde butonul pentru refresh* toggleFab();  
 **return**;  
 }  
 // Uneori fragmentul conține foarte multe date, iar ecranul va părea blocat, așadar folosind un Runnable care va încărca fragmentul va apărea un cross fade effect

Runnable mPendingRunnable = **new** Runnable() {  
 @Override  
 **public void** run() {  
 *// update the main content by replacing fragments* Fragment fragment = getHomeFragment();  
 FragmentTransaction fragmentTransaction = getSupportFragmentManager().beginTransaction();  
 fragmentTransaction.setCustomAnimations(android.R.anim.***fade\_in***,  
 android.R.anim.***fade\_out***);  
 fragmentTransaction.replace(R.id.***frame***, fragment, *CURRENT\_TAG*);  
 fragmentTransaction.commitAllowingStateLoss();  
 }  
 };  
  
 *// Dacă mPendingRunnable nu este null, adaugă-l în coadă.* **if** (mPendingRunnable != **null**) {  
 **mHandler**.post(mPendingRunnable);  
 }  
  
 *// Afișează sau ascunde butonul pentru refresh* toggleFab();  
  
 *//Closing drawer on item click* **drawer**.closeDrawers();  
  
 *// refresh toolbar menu* invalidateOptionsMenu();  
}

Metoda getHomeFragment() va returna pe baza navItemIndex un fragment care ulterior va fi încărcat apoi în FrameLayout –ul din app\_bar\_main.xml

**private** Fragment getHomeFragment() {  
 **switch** (*navItemIndex*) {  
 **case** 0:  
 *// home* FeedFragment homeFragment = FeedFragment.*newInstance*(*user*);  
 **return** homeFragment;  
 **case** 1:  
 *// new post* CameraFragment photosFragment = CameraFragment.*newInstance*(*user*);  
 **return** photosFragment;  
 **case** 2:  
 *// gallery* ProfileFragment profileFragment = ProfileFragment.*newInstance*(*user*, **target\_user**);  
 **return** profileFragment;  
 **case** 3:  
 *// settings* SettingsFragment settingsFragment = SettingsFragment.*newInstance*(*user*);  
 **return** settingsFragment;  
 **default**:  
 **return** FeedFragment.*newInstance*(*user*);  
 }  
}

Fragmentele vor fi lansate folosind metoda statică newInstance, deoarece în cazul fiecărui fragment se vor pasa anumite informații.

În cazul în care utilizatorul nu este autentificat sau se apasă pe opțiunea Logout din navigation menu se va apela metoda logoutUser() care va șterge toți utilizatorii din tabela users, va opri toate requesturile către sever și va lansa activitatea de autentificare.

**private void** logoutUser() {  
 **session**.setLogin(**false**);  
  
 *db*.deleteUsers();  
 Log.*d*(***TAG***,**"User is not logged in anymore"**);  
 *// Launching the login activity* Intent intent = **new** Intent(MainActivity.**this**, LoginActivity.**class**);  
 startActivity(intent);  
 AppController.*getInstance*().getRequestQueue().cancelAll(**"get\_feed"**);  
 finish();  
}

1. **LoginActivity**

Activitate care va prelua informațiile de la utilizator: email și parola și va transmite un request de autentificare către sever folosind HTTP\_POST prin intermediul librăriei Volley, comunicația fiind asincronă, fiecare request se va adăuga într-o coadă, între timp UI-ul nu rămâne blocat.

Layoutul activități fiind alcătuit din 2 EditText-uri email și parolă, un Imageview pentru logo-ul aplicației, 3 butoane: logare, logare folosind facebook, înregistrare.

Pentru SDK-ul Facebook am adăugat în build.gradle

implementation **'com.facebook.android:facebook-android-sdk:[4,5)'**

Pentru a folosi Volley am adăugat în build.gradle   
implementation **'com.mcxiaoke.volley:library-aar:1.0.0'**

<**RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:app="http://schemas.android.com/apk/res-auto"  
 xmlns:tools="http://schemas.android.com/tools"  
 android:layout\_width="fill\_parent"  
 android:layout\_height="fill\_parent"  
 android:background="@drawable/gradient"  
 android:orientation="vertical"  
 tools:context=".LoginActivity"**>  
 <**LinearLayout  
 android:layout\_width="fill\_parent"  
 android:layout\_height="wrap\_content"  
 android:layout\_gravity="center"  
 android:orientation="vertical"  
 android:paddingLeft="20dp"  
 android:paddingRight="20dp"**>  
 <**RelativeLayout  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:layout\_weight=".5"  
 android:paddingTop="20dp"**>  
 <**ImageView  
 android:id="@+id/imageView"  
 android:layout\_width="match\_parent"  
 android:layout\_height="200dp"  
 android:paddingStart="20dp"  
 android:paddingTop="20dp"  
 android:scaleType="centerCrop"  
 android:src="@drawable/logo"  
 tools:layout\_editor\_absoluteX="86dp"** />  
 </**RelativeLayout**>  
 <**EditText  
 android:id="@+id/loginEmail"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:ems="10"  
 android:hint="@string/LoginEmailHint"  
 android:inputType="textEmailAddress"  
 tools:layout\_editor\_absoluteX="84dp"  
 tools:layout\_editor\_absoluteY="240dp"** />  
 <**EditText  
 android:id="@+id/loginPassword"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:layout\_weight="0"  
 android:ems="10"  
 android:hint="@string/LoginPasswordHint"  
 android:inputType="textPassword"  
 tools:layout\_editor\_absoluteX="85dp"  
 tools:layout\_editor\_absoluteY="285dp"** />  
 <**Button  
 android:id="@+id/loginButton"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:text="@string/LoginButton"  
 tools:layout\_editor\_absoluteX="148dp"  
 tools:layout\_editor\_absoluteY="372dp"** />  
 <**com.facebook.login.widget.LoginButton  
 android:id="@+id/login\_button"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:layout\_gravity="center\_horizontal"  
 android:layout\_marginTop="30dp"  
 android:layout\_marginBottom="30dp"** />  
 <**Button  
 android:id="@+id/loginNoAccountButton"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:background="@android:color/transparent"  
 android:text="@string/LoginNoAccount"  
 tools:layout\_editor\_absoluteX="85dp"  
 tools:layout\_editor\_absoluteY="494dp"** />  
  
 </**LinearLayout**>  
</**RelativeLayout**>

Clasa LoginActivity are următorii membrii:

**public class** LoginActivity **extends** Activity {  
 **private static final** String ***TAG*** = LoginActivity.**class**.getSimpleName();  
 **private** Button **btnLogin**;  
 **private** Button **btnLinkToRegister**;  
 **private** EditText **inputEmail**;  
 **private** EditText **inputPassword**;  
 **private** ProgressDialog **pDialog**;  
 **private** SessionManager **session**;  
 **private** SQLiteHandler **db**;  
 **private** CallbackManager **callbackManager**;  
 **private** LoginButton **mButtonLogin**;  
 **private** User **user**;

unde pDialog este folosit pentru afișarea unui dialog cât se așteaptă răspunsul de la server, callbackManager și mButtonLogin sunt folosite pentru implementarea autentificării folosind facebook, metoda onCreate fiind:

@Override  
**public void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(R.layout.***activity\_login***);  
 **mButtonLogin** = (LoginButton) findViewById(R.id.***login\_button***);

// Setez permisiunle care vor fi cerute utilizatorului facebook   
 **mButtonLogin**.setReadPermissions(Arrays.*asList*(  
 **"public\_profile"**, **"email"**, **"user\_birthday"**,**"user\_photos"**,**"user\_location"**,**"user\_gender"**));  
 **callbackManager** = CallbackManager.Factory.*create*();  
  
 *// Callback registration* **mButtonLogin**.registerCallback(**callbackManager**, **new** FacebookCallback<LoginResult>() {  
 @Override  
 **public void** onSuccess(LoginResult loginResult) {  
 *// App code* GraphRequest request = GraphRequest.*newMeRequest*(  
 loginResult.getAccessToken(),  
 **new** GraphRequest.GraphJSONObjectCallback() {  
 @Override  
 **public void** onCompleted(JSONObject object, GraphResponse response) {  
 Log.*v*(**"LoginActivity"**, response.toString());  
 **try** {  
 *// Application code* String email = object.getString(**"email"**);  
 **int** id = object.getInt(**"id"**);  
 String fullname = object.getString(**"name"**);  
 String birthday = object.getString(**"birthday"**);  
 String loc = object.getString(**"location"**);  
 String profileImage = object.getString(**"profile\_pic"**);  
 String gender = object.getString(**"gender"**);  
 **user** = **new** User(id, fullname, email, profileImage, birthday, loc, gender, 0, **"d"**,**"14-12-1995"**,**"none"**);  
 checkLogin\_Social();  
 } **catch** (JSONException e) {  
  
 }  
 }  
 });  
 Bundle parameters = **new** Bundle();  
 parameters.putString(**"fields"**, **"id,name,email,gender,birthday,location,photos{images}"**);  
 request.setParameters(parameters);  
 request.executeAsync();  
  
 }  
  
 @Override  
 **public void** onCancel() {  
 *// App code* Log.*v*(**"LoginActivity"**, **"cancel"**);  
 }  
  
 @Override  
 **public void** onError(FacebookException exception) {  
 *// App code* Log.*v*(**"LoginActivity"**, exception.getCause().toString());  
 }  
 });  
  
  
 **inputEmail** = (EditText) findViewById(R.id.***loginEmail***);  
 **inputPassword** = (EditText) findViewById(R.id.***loginPassword***);  
 **btnLogin** = (Button) findViewById(R.id.***loginButton***);  
 **btnLinkToRegister** = (Button) findViewById(R.id.***loginNoAccountButton***);  
  
 *// Progress dialog* **pDialog** = **new** ProgressDialog(**this**);  
 **pDialog**.setCancelable(**false**);  
  
 *// SQLite database handler* **db** = **new** SQLiteHandler(getApplicationContext());  
  
 *// Session manager* **session** = **new** SessionManager(getApplicationContext());  
  
 *// Verific dacă utilizatorul este cumva autentificat* **if** (**session**.isLoggedIn()) {  
 *// Utilizatorul este deja autentificat, redirecționare către MainActivity* Intent intent = **new** Intent(LoginActivity.**this**, MainActivity.**class**);  
 startActivity(intent);  
 finish();  
 }  
  
 *// Login button Click Event* **btnLogin**.setOnClickListener(**new** View.OnClickListener() {  
  
 **public void** onClick(View view) {  
 String email = **inputEmail**.getText().toString().trim();  
 String password = **inputPassword**.getText().toString().trim();  
  
 *// Verific dacă s-au introdus date în cele 2 EditText-uri* **if** (!email.isEmpty() && !password.isEmpty()) {  
 *// login user* checkLogin(email, password);  
 } **else** {  
 *// Afișare mesaj* Toast.*makeText*(getApplicationContext(),  
 **"Please enter the credentials!"**, Toast.***LENGTH\_LONG***)  
 .show();  
 }  
 }  
  
 });  
  
 *// Redirecționare către activitatea de înregistrare* **btnLinkToRegister**.setOnClickListener(**new** View.OnClickListener() {  
  
 **public void** onClick(View view) {  
 Intent i = **new** Intent(getApplicationContext(),  
 RegisterActivity.**class**);  
 startActivity(i);  
 finish();  
 }  
 });  
  
}

Metoda checkLogin va trimite requestul către URL\_LOGIN folosind metoda HTTP\_POST cu următoarele informații.

**private void** checkLogin(**final** String email, **final** String password) {  
 *// Tag used to cancel the request* String tag\_string\_req = **"req\_login"**;  
  
 **pDialog**.setMessage(**"Logging in ..."**);  
 showDialog();  
  
 StringRequest strReq = **new** StringRequest(Method.***POST***,  
 AppConfig.*URL\_LOGIN*, **new** Response.Listener<String>() {  
  
 @Override  
 **public void** onResponse(String response) {  
 Log.*d*(***TAG***, **"Login Response: "** + response.toString());  
 hideDialog();  
  
 **try** {  
 JSONObject jObj = **new** JSONObject(response);  
 **boolean** error = jObj.getBoolean(**"error"**);  
  
*// Verific daca cumva a fost o eroare* **if** (!error) {  
 **session**.setLogin(**true**);  
 JSONObject jSUser = jObj.getJSONObject(**"user"**);

//Creez un utilizator cu datele din JSON  
 **user** = **new** User(jSUser.getInt(**"id"**),jSUser.getString(**"fullname"**),  
 jSUser.getString(**"email"**), jSUser.getString(**"profilePicture"**),  
 jSUser.getString(**"birthday"**), jSUser.getString(**"country"**),jSUser.getString(**"gender"**),jSUser.getInt(**"points"**), jSUser.getString(**"use\_Recognizer"**), jSUser.getString(**"created\_at"**), jSUser.getString(**"about"**));**user**.**numberofFriends**.set(jSUser.getLong(**"nr\_friends"**));  
 **user**.**numberofAppereances**.set(jSUser.getLong(**"nr\_foundIn"**));  
 **user**.**numberofPhotosTaken**.set(jSUser.getLong(**"nr\_pictures"**));  
 // Inserez userul in tabelul users  
 **db**.addUser(**user**);  
  
 *// Lansez activitatea main* Intent intent = **new** Intent(LoginActivity.**this**,  
 MainActivity.**class**);  
 startActivity(intent);  
 finish();  
 } **else** {  
 *// Eroare in autentificare* String errorMsg = jObj.getString(**"error\_msg"**);  
 Toast.*makeText*(getApplicationContext(),  
 errorMsg, Toast.***LENGTH\_LONG***).show();  
 }  
 } **catch** (JSONException e) {  
 *// JSON error* e.printStackTrace();  
 Toast.*makeText*(getApplicationContext(), **"Json error: "** + e.getMessage(), Toast.***LENGTH\_LONG***).show();  
 }  
  
 }  
 }, **new** Response.ErrorListener() {  
  
 @Override  
 **public void** onErrorResponse(VolleyError error) {  
 Log.*e*(***TAG***, **"Login Error: "** + error.getMessage());  
 Toast.*makeText*(getApplicationContext(),  
 error.getMessage(), Toast.***LENGTH\_LONG***).show();  
 hideDialog();  
 }  
 }) {  
  
 @Override  
 **protected** Map<String, String> getParams() {  
 *// parametrii pentru URL\_LOGIN* Map<String, String> params = **new** HashMap<String, String>();  
 params.put(**"email"**, email);  
 params.put(**"password"**, password);  
  
 **return** params;  
 }  
  
 };  
  
 *// Adaug requestul în coadă* AppController.*getInstance*().addToRequestQueue(strReq, tag\_string\_req);  
 }

1. **RegisterActivity**

Rolul acestei activități este de a culege informațiile de la utilizator, a le valida, iar ulterior a le trimite ca și request către server care apoi va finaliza înregistrarea. Layoutul activității conține: ScrollView ( deoarece sunt foarte multe elemente și va fi necesară scrolling-ul printre ele ), ImageView pentru logo, 5 EditText-uri: nume, parolă, confirmare parolă, data de naștere, email, Radio Grup pentru selecția sexului, un CountryPicker pentru selectarea țării și 2 butoane.

Pentru a folosii CountryPicker-ul a fost integrată librăria în build.gradle

implementation **'com.hbb20:ccp:2.2.0'**

Se setează acțiune onClick în cazul butonului btnRegister, care dacă toate inputurile sunt prezente și valide va apela metoda: registerUser

**btnRegister**.setOnClickListener(**new** View.OnClickListener() {  
 **public void** onClick(View view) {  
 String name = **inputFullName**.getText().toString().trim();  
 String email = **inputEmail**.getText().toString().trim();  
 String password = **inputPassword**.getText().toString().trim();  
 String conf\_password = **confirmPassword**.getText().toString().trim();  
 String date = **inputDate**.getText().toString().trim();  
 String country = **countryCodePicker**.getSelectedCountryName();  
 RadioButton genderSelector = (RadioButton) findViewById(**selectedGender**.getCheckedRadioButtonId());  
 String gender = genderSelector.getText().toString();  
 **if**(gender.contains(**"Male"**))  
 {  
 gender = **"m"**;  
 }  
 **else** {  
 gender = **"f"**;  
 }

// Dacă toate inputurile sunt completate  
 **if** (!name.isEmpty() && !email.isEmpty() && !password.isEmpty() && !conf\_password.isEmpty() && !date.isEmpty() && !country.isEmpty()) {

//Dacă adresa de email nu corespunde regex-ului pentru mail  
 **if** (!*isEmailValid*(email)) {  
 Toast.*makeText*(getApplicationContext(),  
 **"Wrong email adress!"**, Toast.***LENGTH\_LONG***)  
 .show();  
 // Dacă parola nu este confirmată cu succes  
 } **else if** (!password.equals(conf\_password)) {  
 Toast.*makeText*(getApplicationContext(),  
 **"The passwords doesn't match!"**, Toast.***LENGTH\_LONG***)  
 .show();  
 // Dacă parola nu corespunde regex-ului pentru parolă   
 }**else if**(!*isPasswordStrong*(password)) {  
 Toast.*makeText*(getApplicationContext(),  
 **"The password should be at least 8 characters, 1 uppercase letter, 1 special character"**, Toast.***LENGTH\_LONG***)  
 .show();

// Dacă data de naștere nu coincide tiparului: an.LUNĂ.zi   
 }**else if** (!*isValidDate*(date)) {  
 Toast.*makeText*(getApplicationContext(),  
 **"Date must respect pattern: yyyy.MM.dd"**, Toast.***LENGTH\_LONG***)  
 .show();  
 } **else** {

// Dacă toate condițiile au fost împlinite  
 registerUser(name, email, password, date, gender, country);  
  
 }  
 }**else** {  
 Toast.*makeText*(getApplicationContext(),  
 **"Please enter your details!"**, Toast.***LENGTH\_LONG***)  
 .show();  
 }  
 }  
});

Metoda registerUser este similară cu cea checkLogin însă diferă prin implementarea onResponse și getParams, iar StringRequestul va fi transmis către URL\_REGISTER în acest caz.

*// Tag used to cancel the request*String tag\_string\_req = **"req\_register"**;  
  
**pDialog**.setMessage(**"Registering ..."**);  
showDialog();  
  
StringRequest strReq = **new** StringRequest(Method.***POST***,  
 AppConfig.*URL\_REGISTER*, **new** Response.Listener<String>() {  
  
 @Override  
 **public void** onResponse(String response) {  
 Log.*d*(***TAG***, **"Register Response: "** + response.toString());  
 hideDialog();  
 **try** {  
 JSONObject jObj = **new** JSONObject(response);  
 **boolean** error = jObj.getBoolean(**"error"**);  
 **if** (!error) {  
  
 Toast.*makeText*(getApplicationContext(), **"User successfully registered. Try login now!"**, Toast.***LENGTH\_LONG***).show();  
  
 *// Launch login activity* Intent intent = **new** Intent(  
 RegisterActivity.**this**,  
 LoginActivity.**class**);  
 startActivity(intent);  
 finish();  
 }  
 **else** {  
  
 *// Error occurred in registration. Get the error  
 // message* String errorMsg = **"Something went wrong! Please try again!"**;  
 Toast.*makeText*(getApplicationContext(),  
 errorMsg, Toast.***LENGTH\_LONG***).show();  
 }  
 } **catch** (JSONException e) {  
 e.printStackTrace();  
 }  
  
}

@Override  
 **protected** Map<String, String> getParams() {  
 *// Posting params to register url* Map<String, String> params = **new** HashMap<String, String>();  
 params.put(**"fullname"**, name);  
 params.put(**"email"**, email);  
 params.put(**"password"**, password);  
 params.put(**"country"**, country);  
 params.put(**"gender"**, gender);  
 params.put(**"birthday"**, birthday);  
 params.put(**"profilePicId"**, **"1"**);  
 params.put(**"useRecognizer"**, **"n"**);  
 **return** params;  
 }  
  
};

Metodele de validare returnează true dacă sunt intrările sunt valide, altfel vor returna false.

**public static boolean** isPasswordStrong(String password) {  
 **return *VALID\_PASSWORD\_REGEX*** .matcher(password).matches();  
}

**private static boolean** isEmailValid(String email) {  
 **return *VALID\_EMAIL\_ADDRESS\_REGEX*** .matcher(email).matches();  
}

**private static boolean** isValidDate(String inDate) {  
 SimpleDateFormat dateFormat = **new** SimpleDateFormat(**"yyyy.MM.dd"**);  
 dateFormat.setLenient(**false**);  
 **try** {  
 dateFormat.parse(inDate.trim());  
 } **catch** (ParseException pe) {  
 **return false**;  
 }  
 **return true**;  
}

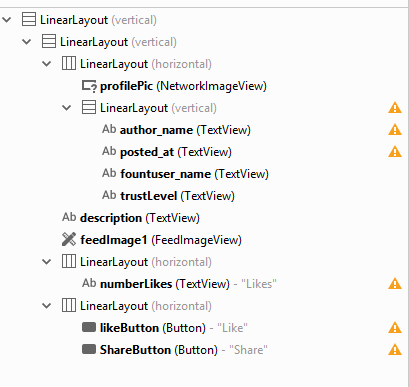
1. **FeedFragment**

Are ca rol preluarea informațiilor care urmează să fie afișate în feed-ul unui utilizatorului. Obiectul de tip users este pasat fragmentului prin metoda newInstance. Informațiile vor fi returnate în urma apelului scriptului feeds.php care primește folosind metoda HTTP.POST id-ul utilizatorului.

Pentru fiecare element din interiorul array-ului JSON returnat de către scriptul PHP se creează un obiect de tip FeedItem care va conține toate informațiile despre obiectul Feed.

**public class** FeedItem {  
 **private int id**, **likes**, **trustLevel**;  
 **private boolean isLiked**;  
 **private** String **author\_name**, **description**, **found\_user**, **picture\_url**, **profile\_picture\_url**, **posted\_at**;  
 **private int author\_id**, **founduser\_id**;

Iar apoi folosind un BaseAdapter personalizat sub numele FeedListAdapter care va primi ca parametrii activitatea, lista de elemente de tip feed și utilizatorul și va seta conținutul elementelor UI pentru fiecare obiect Feed Item creeat care va avea layoutul sub forma:



Unde likeButton va apela scriptul actions.php căruia îi va pasa id-ul utilizatorului, id-ul elementului din feed și o comandă, iar scriptul va insera sau șterge în funcție de comanda oferită informația din likedFeed.

Informația privind conținutul elementelor din feed, mai exact rezultatul scriptului feeds.php se va salva în memoria cache, iar la lansarea fragmentului dacă există în memorie informații de la acel tag, nu se va trimite un request nou, decât la refresh-ul apăsat de către utilizator, apelându-se astfel metoda: getFeeds() care implementează requestul către server.

Cache cache = AppController.*getInstance*().getRequestQueue().getCache();  
Entry entry = cache.get(*URL\_FEED*);  
**if** (entry != **null**) {  
 *// fetch the data from cache* **try** {  
 String data = **new** String(entry.**data**, **"UTF-8"**);  
 **try** {  
 *parseJsonFeed*(**new** JSONObject(data));  
 } **catch** (JSONException e) {  
 e.printStackTrace();  
 }  
 } **catch** (UnsupportedEncodingException e) {  
 e.printStackTrace();  
 }  
  
} **else** {  
 *getFeeds*();  
}

Metoda parseJsonFeed are ca scop despachetarea informațiilor recepționate de la server, creearea listei de elemente de tip feed care vor fi atașate adaptorului, iar apoi notificarea acestuia pentru actualizarea UI-ului.

**private static void** parseJsonFeed(JSONObject response) {  
 **try** {  
 JSONArray feedArray = response.getJSONArray(**"feed"**);  
 Log.*d*(***TAG***, **"AICI: "**+response.toString());  
 **for** (**int** i = 0; i < feedArray.length(); i++) {  
  
 JSONObject feedObj = (JSONObject) feedArray.get(i);  
 FeedItem item;  
 **if**(feedObj.getString(**"fullname"**).isEmpty()) {  
 item = **new** FeedItem(feedObj.getInt(**"id"**), feedObj.getString(**"author\_name"**),  
 feedObj.getString(**" description"**),  
 **"none"**, feedObj.getString(**"picture\_name"**),  
 feedObj.getString(**"profile\_pic"**),  
 feedObj.getString(**"posted\_at"**), feedObj.getInt(**"LIKES"**),  
 -1, feedObj.getInt(**"LIKED"**), feedObj.getInt(**"author\_id"**),-1);  
 }  
 **else** {  
 item = **new** FeedItem(feedObj.getInt(**"id"**), feedObj.getString(**"author\_name"**),  
 feedObj.getString(**"description"**),  
 feedObj.getString(**"fullname"**), feedObj.getString(**"picture\_name"**),  
 feedObj.getString(**"profile\_pic"**),  
 feedObj.getString(**"posted\_at"**), feedObj.getInt(**"LIKES"**),  
 feedObj.getInt(**"trust"**), feedObj.getInt(**"LIKED"**),feedObj.getInt(**"author\_id"**),feedObj.getInt(**"founduser\_id"**));  
 }  
 Log.*d*(***TAG***,**"Sunt aici, post\_pic: "**+item.getPicture\_url()+**" si profile\_pic:"**+item.getProfile\_picture\_url());  
 *feedItems*.add(item);  
 }  
  
 *// notify data changes to list adapater  
 listAdapter*.notifyDataSetChanged();  
 } **catch** (JSONException e) {  
 e.printStackTrace();  
 }  
}

1. **ProfileFragment**

Spre deosebire de FeedFragment care primea informațiile utilizatorul aplicației, ProfileFragment mai primește și informații cu privire la utilizatorul al cărui profil urmează a fi afișat, mai exact id-ul acestuia.

**public static** ProfileFragment newInstance(User user, **int** id) {  
 ProfileFragment fragmentFirst = **new** ProfileFragment();  
 Bundle args = **new** Bundle();  
 args.putSerializable(**"user"**,user);  
 args.putInt(**"userId"**,id);  
 fragmentFirst.setArguments(args);  
 **return** fragmentFirst;  
}

Pe baza acestor informații se va apela metoda getPosts(target\_id) care va obține în urma apelării scriptului PHP gallery.php un JSON array unde fiecare element va reprezenta un element din profil reprezentat de un obiect ProfileItem.

**public class** ProfileItem **extends** BaseObservable {  
 String **imageUrl**;  
 **int imageId**;  
  
  
 @BindingAdapter(**"imageUrl"**)  
 **public static void** loadImage(ImageView view, String imageUrl) {  
 Glide.*with*(view.getContext())  
 .load(imageUrl)  
 .into(view);  
 }

În cazul Profilului este necesară prezența unui adaptor, numit PostsAdapter, numai pentru afișarea imaginilor, informațiile utilizatorului care vor fi afișate în elemente UI se va face folosind librăria databinding.

În urma apelării scriptului gallery.php răspunsul va fi sub forma:

**private void** getPosts(**final int** id) {  
 *posts*.clear();  
 StringRequest strReq = **new** StringRequest(Request.Method.***POST***,  
 AppConfig.*URL\_GALLERY*, **new** Response.Listener<String>() {  
  
 @Override  
 **public void** onResponse(String response) {  
 Log.*d*(***TAG***, **"Login Response: "** + response.toString());  
 **try** {  
 JSONObject jObj = **new** JSONObject(response);  
 **boolean** error = jObj.getBoolean(**"error"**);  
  
*// Check for error node in json* **if** (!error) {  
 parseJsonProfile(jObj, id);  
 JSONObject jSUser = jObj.getJSONObject(**"user"**);  
 **user**.setName(jSUser.getString(**"fullname"**));  
 **user**.setBirthday(jSUser.getString(**"birthday"**));  
 **user**.setCountry(jSUser.getString(**"country"**));  
 **user**.setProfileImage(jSUser.getString(**"profilePicture"**));  
 **user**.setAbout(jSUser.getString(**"about"**));  
 **user**.**numberofFriends**.set(jSUser.getLong(**"nr\_friends"**));  
 **user**.**numberofAppereances**.set(jSUser.getLong(**"nr\_foundIn"**));  
 **user**.**numberofPhotosTaken**.set(jSUser.getLong(**"nr\_pictures"**));

Metoda parseJsonProfile este similară cu metoda parseJsonFeed prezentată anterior însă diferă prin faptul că sunt creeate elemente de tip ProfileItem care vor fi apoi pasate printr-o listă posts adaptorului care va adăuga pozele în profil: PostsAdapter

ProfileItem item = **new** ProfileItem();  
item.setImageUrl(AppConfig.*URL\_SERVER*+feedObj.getString(**"pic\_name"**));  
item.setimageId(Integer.*parseInt*(feedObj.getString(**"pic\_id"**)));  
Log.*d*(***TAG***, **"Am link:"**+AppConfig.*URL\_SERVER*+feedObj.getString(**"pic\_name"**));  
*posts*.add(item);

Iar în cazul în care o poză din profil va fi apăsată de către utilizatorul al cărui profil este afișat acea poză va defeni fotografia de profil a acestuia:

@Override  
**public void** onPostClicked(ProfileItem post) {

**//Daca utilizatorul care folosește aplicația (user.getId()**

**//Este si utilizatorul al carui profil este afișat targetuser\_id**   
 **if** (**user**.getId() == **targetuser\_id**) {  
 **user**.setProfileImage(post.getImageUrl().substring(AppConfig.*URL\_SERVER*.length() - 1));

**// Modific imaginea si in NavigationViewer**  MainActivity.*modifyUser*(**user**);

**// Modific si in tabelul users de pe server informația privind id-ul pozei de profil**   
 modifyProfilePic(**user**.getId(), post.getImageId());  
  
  
 NavigationView navigationView = (NavigationView) getActivity().findViewById(R.id.***nav\_view***);  
 View navHeader = navigationView.getHeaderView(0);  
 ImageView imgProfile = (ImageView) navHeader.findViewById(R.id.***img\_profile***);  
 *// Actualizez imaginea din profil* Glide.*with*(**this**).load(**user**.getProfileImage())  
 .crossFade()  
 .thumbnail(0.5f)  
 .bitmapTransform(**new** CircleTransform(getActivity().getApplicationContext()))  
 .diskCacheStrategy(DiskCacheStrategy.***ALL***)  
 .into(imgProfile);  
 Toast.*makeText*(getActivity().getApplicationContext(), **"Profile picture updated!"**, Toast.***LENGTH\_SHORT***).show();  
 }  
}

Pe lângă acestea layoutul profilului este prevăzut cu un buton de download images care va fi vizibil numai dacă targetUser\_id (id-ul utilizatorul al cărui profil este afișat ) este identic cu user.getid() ( id-ul utilizatorului care foloseștea aplicația ).

1. **CameraFragment**

# 4 Referințe bibliografice

1. Watson, D., & Friend, R. (1969). Measurement of social-evaluative anxiety. Journal of Consulting and Clinical Psychology, 33(4), 448-457.
2. Harvard Medical School, 2007. National Comorbidity Survey (NCS). (2017, August 21)
3. <https://source.android.com/setup/start/build-numbers> Codenames, Tags and Build Numbers
4. Ed Burnette. Hello, Android: Introducing Google’s Mobile Development Platform, Third Edition
5. Caifeng Shan and Tommaso Gritti Philips Research, High Tech Campus 36, Eindhoven 5656 AE, The Netherlands. Learning Discriminative LBP-Histogram Bins for Facial Expression Recognition
6. Ojala : Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. IEEE Transactions on Pattern Analysis and Machine Intelligence 24 (2002) 971–987
7. Ojala : A comparative study of texture measures with classification based on feature distributions. Pattern Recognition 29 (1996) 51–59
8. M. Welling, „Fisher Linear Discriminant Analysis,” Toronto.
9. Martinez, A. M.; Kak, A. C. (2001). ["PCA versus LDA"](http://www.ece.osu.edu/~aleix/pami01.pdf)  [IEEE Transactions on Pattern Analysis and Machine Intelligence](https://en.wikipedia.org/wiki/IEEE_Transactions_on_Pattern_Analysis_and_Machine_Intelligence) 228–233.
10. Bob Dowling, Jon Warbrick University of Cambridge Computing Service. Web Server Management: Running Apache 2.2 under Linux
11. O’Reilly: Ken Coar & Rich Bowen. Apache Cookbook
12. ["JavaOne 2013 Review: Java Takes on the Internet of Things"](http://www.oracle.com/technetwork/articles/java/afterglow2013-2030343.html). [www.oracle.com](http://www.oracle.com).
13. Herbert Schildt. Java The Complete Reference 10-25
14. [Is Python a good language for beginning programmers?"](https://docs.python.org/faq/general.html#is-python-a-good-language-for-beginning-programmers). General Python FAQ. Python Software Foundation. Retrieved 21 March 2007.
15. Robin Nixon. Learning PHP, MySQL & JavaScript. 35-50
16. M. Negri, G. Pelagatti, L. Sbattella (1989) [Semantics and problems of universal quantification in SQL](http://portal.acm.org/citation.cfm?id=63224.68822&coll=GUIDE&dl=GUIDE)
17. TutorialsPoint. SQL i
18. ["XML 1.0 Origin and Goals"](http://www.w3.org/TR/REC-xml/#sec-origin-goals)*. W3.org*. 14 August 2016*.*
19. Mail::RFC822::Address: regexp-based address validation [Paul Warren](mailto:pdw@ex-parrot.com) 17/09/2012