SUSTAV ZA DETEKCIJU LICA SA UPOZORENJEM NA MOBILNOM UREĐAJU

Vegh, Mario

VISOKO UČILIŠTE ALGEBRA

ZAVRŠNI RAD

SUSTAV ZA DETEKCIJU LICA SA UPOZORENJEM NA MOBILNOM UREĐAJU

Mario Vegh

Zagreb, prosinac 2017.

Student vlastoručno potpisuje završni rad na prvoj stranici ispred predgovora s datumom i oznakom mjesta završetka rada te naznakom:

„Pod punom odgovornošću pisano potvrđujem da je ovo moj autorski rad čiji niti jedan dio nije nastao kopiranjem ili plagiranjem tuđeg sadržaja. Prilikom izrade rada koristio sam tuđe materijale navedene u popisu literature, ali nisam kopirao niti jedan njihov dio, osim citata za koje sam naveo autora i izvor, te ih jasno označio znakovima navodnika. U slučaju da se u bilo kojem trenutku dokaže suprotno, spreman sam snositi sve posljedice uključivo i poništenje javne isprave stečene dijelom i na temelju ovoga rada.“

U Zagrebu, 15.12.2017.

Predgovor

Zahvaljujem mentoru dipl. ing. Bojanu Fulanoviću, na pomoći tijekom izrade završnog rada.

Prilikom uvezivanja rada, Umjesto ove stranice ne zaboravite umetnuti original potvrde o prihvaćanju teme završnog rada kojeg ste preuzeli u studentskoj referadi

Sažetak

Ovaj rad prikazuje cjelovito rješenje praćenja dolaznosti studenata na nastavu. Student pri ulasku u učionicu napravi prepoznavanje lica te mu u realnom vremenu pristiže obavijest na mobitel da je uspješno prisustvovao predavanju. Svrha praktičnog rada je suzbijanje lažnih potpisa od strane kolega, a isto tako i pomoć studentu da nadoknadi predavanja ili vježbe na koje nije stigao doći. Ovaj rad sastoji se od Android aplikacije, Python desktop aplikacije, Python konzolne aplikacije i Php web servisa za komunikaciju putem obavijesti. Android aplikaciju koriste studenti, profesori i administratori za pregled rasporeda, grupa i kolegija. Python desktop aplikacija koristi se za prepoznavanje studenta i registriranje dolaznosti svakog studenta. Konzolna aplikacija u Pythonu koristi se kao kreator rasporeda za svaki dan i traženje dodatnih termina za svakog studenta u slučaju da nije prisustvovao predavanjima. Php web servisi koriste se za obavještavanje korisnika o dolaznosti na predavanje ili o obavijesti vezanoj za kolegij. Baza podataka za komunikaciju između Python aplikacije, Android aplikacije i Php aplikacije je Firebase. Kod Php aplikacije se još koristi dodatna baza MySQL za pohranu identifikacijske oznake uređaja koji se spojio na Firebase.

Ključne riječi: Android, Python, Php, Firebase, MySQL

Summary

This project shows complete solution on tracking down students' attendance in their courses. The student, on entering the classroom, takes part in a facial recognition process, after which they receive a notification confirming their attendance in real time on their cellphone. The purpose of this project is to reduce number of fake signatures amongst classmates, aswell as providing help to the student in catching up with missed classes and practical exercises. This project consist of an Android application, a Python desktop application, a Python console application and a PHP web services for communication via notifications. The Android application is used by students, professors and administrators that overview schedules, groups and courses. The Python desktop application is used for facial recognition and tracking attendance for each student. The Python console application is used as a tool to create schedules for each day and also finding additional appointments for each student in case they missed the class. The PHP web services is used to notify the user about attendance or similar notifications tied to their courses. The datababase for communication between the Python desktop application, Android application and the PHP application is Firebase. PHP application uses MySQL database for saving the ID numbers of devices which are connected to Firebase.

Keywords: Android, Python, Php, Firebase, MySQL

Sadržaj

1. Uvod

Završni rad obrađuje temu praćenja dolazaka studenata na kolegije. Sustav je osmišljen kao alternativa postojećem sustavu. Za praćenje dolazaka studenata na kolegije još uvijek koriste se stari i neučinkoviti principi, a to su papir i prozivanje svakog studenta pojedinačno. Studenti se još uvijek mogu koristiti potpisom kolege kao da su stvarno prisutni na dotičnom predavanju. Sustav za praćenje studenata može se podijeliti u tri segmenta. Prvi segment - Android aplikacija, drugi segment - komunikacija putem obavijesti i treći segment - detekcija i prepoznavanje lica.

Segment Android aplikacija služi kao glavno mjesto za praćenje pojedinog studenta u dolasku na nastavu. Administrator unutar aplikacije kreira korisničke račune za role profesor i student. Svaki student dodijeljen je jednoj grupi kojoj administrator kreira raspored. Profesor unutar aplikacije nadzire prisutnost svakog studenta te ga obavještava u slučaju važnosti nekog događaja. Student unutar aplikacije prati dnevni raspored i prisutnost kolegijima.

Segment komunikacije putem obavijesti koristi se za obavještavanje studenata o događajima vezanim za određeni kolegij. Sastavni dijelovi komunikacije su web servis pisan u PHP programskom jeziku te platforma Firebase za razmjenu poruka u oblaku. Pomoću web servisa prosljeđuje se zahtjev na Firebase koji šalje obavijest na mobilni uređaj.

Segment detekcije i prepoznavanja lica odnosi se na ulazak studenta u učionicu. Sustav detektira i prepoznaje lice određenog studenta te mu pristiže obavijest na mobilni uređaj. Profesor u realnom vremenu na svom mobilnom uređaju vidi prisutnost studenata.

Cilj i svrha aplikacije je pomoć studentima u praćenju prisustva na upisanim kolegijima, a da je pri tom sve transparentno. Ujedno je ovo rješenje pomoć profesoru u bržem upisivanju prisutnih studenata.

2. Arhitektura Android operacijskog sustava

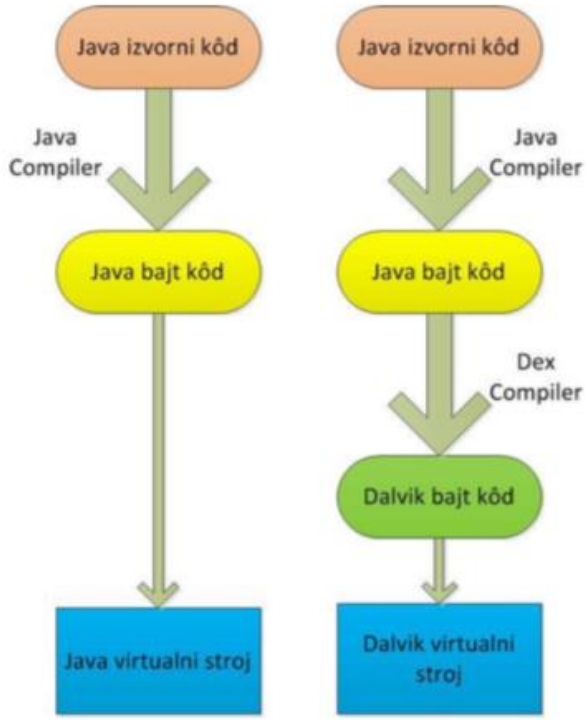
Android operacijski sustav baziran je na Linux 2.6 kernelu i napisan je u C i C++ programskom jeziku. Razlog odabira Linuxa leži u tome što je Linux portabilan, siguran i otvorenog koda (engl. *open source*). Linux je portabilan zato što je napisan u C programskom jeziku te se lako može ugraditi na bilo koji uređaj kao što su mobilni uređaj te pametni sat. Operacijski sustav Linux također je siguran jer se svaka android aplikacija pokreće kao zaseban proces što omogućuje da jedna aplikacija ne može utjecati na drugu osim ako eksplicitno tako ne odredimo[1].

Android također dolazi već s ugrađenim bibliotekama napisanima u programskom jeziku C i C++. Neke od najpoznatiji ugrađenih biblioteka su WebKit, SQLite, OpenGL. Webkit je mrežni pokretač (engl. *web engine*) i koristi ga Safari web preglednik, nekad ga je koristio i Chrome, no Chrome je 2013. prešao na svoj mrežni pokretač (engl. *web engine*) Blink. SQLite je jednostavna ugrađena baza podataka koja koristi 5 vrsta tipova podataka:

- Null – U tablici nemamo podatka
- Integer – Brojevi bez decimala (1, 2, 3)
- Real – Brojevi s decimalnim vrijednostima (10.05, 11.5)
- Text – Običan korisnikov tekst
- Blob – Unos slike u bazu u binarnom obliku

OpenGL se koristi za 3D grafiku na mobilnom uređaju.

U Android aplikaciji kôd pišemo u Javi, a zatim se taj kod pretvara u Java bajtkod (engl. bytecode). Putem *Dex* prevodioca (eng. compiler) prevodi se Java bajtkod (engl. bytecode) u Dalvik bajtkod (engl. bytecode) koji se zatim izvršava na virtualnom stroju Dalvik. Dalvik je virtualni Googlov stroj kojim se rješavaju problemi kao što su trajanje baterije, procesorska moć[1].



Slika 1. Prikaz prevađanja Java kôda

Prikaz prevađanja Java izvornog koda (Slika 1.) u odnosu na prevađanje izvornog kôda kod Android aplikacije. Kod Android aplikacije imamo jedan korak više kako bi se došlo do virtualnog stroja[1].

Unutar ove tablice (Tablica 1.) nabrojane su trenutne verzije Androida kojima Google pruža podršku.

Tablica 1. Prikaz podržanih verzija Androida od strane Googla

Kodno ime	Verzija	Datum izdavanja	API
Lollipop	5.0 – 5.11	12. studeni, 2014.	21 – 22
Marshmallow	6.0 – 6.0.1	5. listopad, 2015.	23
Nougat	7.0 – 7.1.2	22. kolovoz, 2016.	24 – 25
Oreo	8.0 – 8.1	21. kolovoz, 2017.	26 – 27

2.1. Životni ciklus Android aplikacije

Životni ciklus Android aplikacije započinje pokretanjem Android aplikacije. Android aplikacija pokreće se kao zaseban Linux proces što Linux čini sigurnim operacijskim sustavom kako je i navedeno u prethodnom tekstu. Unutar Android aplikacije postoji mehanizam koji se zove upravitelj aktivnosti (engl. *activity manager*) i on upravlja životnim ciklusom aktivnosti[1].

Aktivnost je glavna komponenta svake Android aplikacije i otvara se čim pokrenemo aplikaciju. Tu započinje njen životni ciklus. Pri stvaranju novog Android projekta, unutar tog projekta, dobivamo klasu glavna aktivnost (engl. *main activity*) koja se pokreće prva pri pokretanju aplikacije.

```
<activity android:name=".MainActivity" >
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category
android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

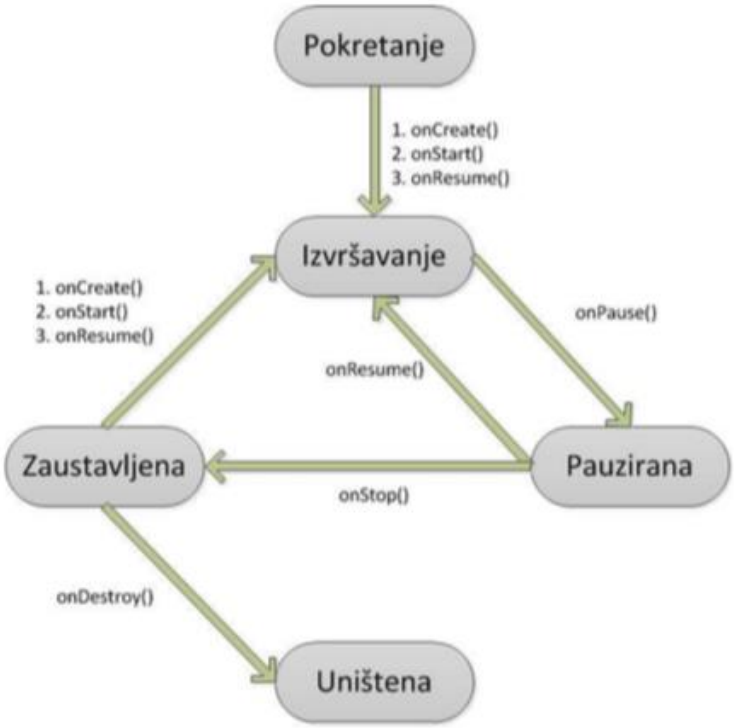
Kôd 1. Prikaz Android manifesta

Unutar Android Manifesta (Kôd 1.) prikazuje se intent filter koji govori operacijskom sustavu Linux koju aktivnost treba pokrenuti prvu pri otvaranju aplikacije.

Pri otvaranju aplikacije pokreće se metoda *onCreate()* i unutar te metode postavljamo datoteku korisničkog sučelja pozivom metode *setContentView()*. Nakon završetka metode *onCreate()* započinje metoda *onStart()*. Unutar metode *onStart()* aplikacija počinje biti vidljiva korisniku, no korisnik ne može unositi sadržaj u aplikaciju. Unutar metode *onResume()* aktivnost je na vrhu stoga te sada korisnik napokon može unijeti svoj željeni tekst u aplikaciju[1].

Kada se korisnik odluči prebaciti na drugu aplikaciju ili isključiti našu aplikaciju tada korisnik ulazi u metodu *onPause()*. Unutar metode *onPause()* možemo sačuvati podatke koje korisnik nije spremio prije izlaska iz aplikacije. Metoda *onStop()* poziva se kada aplikacija više nije vidljiva na ekranu i u tom trenutku oslobađamo resurse, kao na primjer povezanost sa servisom. Posljednja metoda je metoda *onDestroy()* koja se poziva prilikom uklanjanja

aktivnosti iz memorije. Utjecati na metodu *onDestroy()* nije moguće kao s prethodnim metodama jer o ovoj metodi brine se upravitelj aktivnosti (engl. *activity manager*).



Slika 2. Prikaz životnog ciklusa aplikacije

Ova ilustracija (Slika 2.) prikazuje tijek životnog ciklusa aplikacije opisanog iznad. Za pokretanje aplikacije koriste se identične metode izvršavanja kao i kada je aplikacija pokrenuta iz stanje *onStop()*. U slučaju da se iz stanja *onStop()* ne pokrene aplikacija ona se uništava pozivom metode *onDestroy()*. Nakon ponovnog ulaska u aplikaciju iz stanja *onPause()* poziva se metoda *onResume()*. Ukoliko se aplikacija više ne koristi poziva se metoda *onDestroy()*.

3. Platforma Firebase

Firebase je web platforma koja pomaže programerima u izradi njihovih mobilnih i web aplikacija. Uz pomoć Firebase platforme, aplikacije se rade puno brže jer je ugrađeno mnoštvo servisa koji olakšavaju rad programera. Platforma Firebase nastala je 2011. godine u tvrtki Firebase, Inc, a 2014. godine je preuzima Google te ju nastavlja dodatno proširivati s novim mogućnostima. Prvotno je platforma Firebase 2011. godine bila zamišljena kao servis gdje bi ostali ljudi pomoću integriranog API-ja lako mogli složiti svoj vlastiti web za dopisivanje(engl. *chat*). Naposljetku se najviše koristila za spremanje rezultata iz video igara. Sadašnja platforma Firebase broji devetnaest servisa, a posljednji servis je Cloud Firestore[2].

Slika 3. Prikaz servisa iz platforme Firebase

Ova ilustracija (Slika 3.) prikazuje sveukupni set servisa koje je Firebase platforma kreirala. U ovom radu korišteno je nekoliko servisa koje ću u daljnjem tijeku rada detaljno opisati[3].

- Baza u realnom vremenu (engl. *Realtime Database*) – Koristi se kao pozadinski (engl. *backend*) dio bilo mobilne ili web aplikacije i odlična je za sinkronizaciju podataka između raznih platformi, kao što su web, mobilna ili desktop aplikacija.
- Izvješća o greškama (engl. *Crash Reporting*) – Odličan alat za praćenje stanja i izvješća u aplikaciji. U slučaju da je neki korisnik naišao na problem u aplikaciji automatski nas Firebase obavještava o problemu putem naše e-pošte te unutar administracijskog sučelja na platformi.
- Autentifikacija (engl. *Authentication*) – Služi u svrhu prijave korisnika u našu aplikaciju. Korisnik se jednostavno može prijaviti u aplikaciju putem korisničkog računa s neke od društvenih mreža (Facebook, Twitter, Google) ili putem svoje e-pošte.
- Prostor u oblaku (engl. *Cloud Storage*) – Koristi se u svrhu spremanja binarnih zapisa kao na primjer slike, video, zvuk, dokumenti. Vrlo lako prema ranku može se ograničiti pravo na unos binarnog sadržaja iz kojih mapa može čitati sadržaj.
- Prostor na internetu (engl. *Hosting*) – Koristi se za spremanje sadržaja kao na primjer HTML, CSS i JavaScript tipova datoteka.
- Kolekcija u oblaku (engl. *Cloud FireStore*) – Koristi se slično kao baza u realnom vremenu (engl. *RealTime Database*), samo što je baza u realnom vremenu unutar jedne velike kolekcije, dok kod kolekcija u oblaku (engl. *Cloud FireStore*) svaki podčvor čini novu kolekciju unutar baze.
- Google Analitika (engl. *Google Analaytics*) – Služi za praćenje korisnika unutar aplikacije. Uz pomoć ovog dodatka možemo pratiti koje stranice korisnici najviše otvaraju i koliko se vremenski zadržavaju te koje hiperveze su kliknuli.
- Razmjena poruka u oblaku (engl. *Cloud Messaging*) – Uz pomoć ovog servisa možemo poslati našim korisnicima poruke u obliku obavijesti na njihov uređaj te uvijek biti sigurni da će svi korisnici u kratkom roku pogledati našu informaciju koju smo im uputili. Zanimljivost ovog servisa je činjenica da je podržan na više platformi(engl. *cross-platform*) što znači da možemo uputiti poruku putem web-a i pritom će obavijest biti vidljiva na uređajima poput Androida ili iOS-a i obrnuto.
- Reklame na mobitelu (engl. *AdMob*) – Koriste se kako bi razvojni tim mobilne aplikacije uspio nešto i zaraditi putem reklama na mobitelu. Zarada po jednom kliku ovisit će o državi iz koje korisnik pristiže. Maksimalnu zaradu po kliku ćemo ostvariti od državljanina u Sjedinjenim Američkim Državama.
- Oglasi (engl. *AdWords*) putem ovog servisa možemo kreirati kampanju za naš proizvod te doći do ljudi koje zanima naš proizvod. Prikazivanje oglasa na stranicama je u potpunosti besplatno. Kada korisnik klikne na naš oglas tek onda se smanjuju novci koje smo uložili u kampanju.

3.1. Prikaz podatka u Firebase RealTime Database

Firebase RealTime Database je NoSQL baza podataka koja prikazuje podatke u obliku JSON-a. Posebnost ove NoSQL baze je u tome što svi klijenti bilo mobilni ili web dijele istu instancu baza i u realnom vremenu dobivaju nove informacije. U slučaju da nemamo mobilni

promet na našem mobilnom uređaju svi podaci koje smo unosili, spremaju se lokalno na mobilni uređaj. Podaci su lokalno na mobitelu spremljeni u SQLite bazi podataka koju je stvorila instanca Firebase. Kada se mobilni uređaj spoji na Internet istog trena se podaci šalju u bazu i u realnom vremenu svi spojeni klijenti vide naše podatke koje smo poslali[4].

Podaci unutar NoSQL baze podataka strukturirani su u obliku JSON stabla. Prilikom unošenja novog podatka u bazu kreira se novi čvor unutar već postojećeg JSON stabla. Ubačeni podatak može imati identifikacijsku oznaku koju mu programer dodijeli ili je može sama platforma Firebase kreirati. Identifikacijska oznaka koju kreira platforma Firebase sastoji se od brojki i slova i ona ovisi o trenutnom vremenu. Podaci unutar identifikacijske oznake mogu biti u obliku cijelog broja (engl. *integer*), decimalnog broja (engl. decimal number), stanja točnosti (engl. *boolean*), u obliku polja (engl. *array*) te u obliku objekta. U koliko programer kreira vlastitu identifikacijsku oznaku, ona ne smije sadržavati nedozvoljene znakove, kao na primjer točku, ljestve, dolar (., #, $)[4].

Pri spremanju podatka u bazu treba imati na umu što je dublji čvor unutar strukture to će mu više vremena trebati da se prikaže korisniku na ekran. Firebase podržava dubinu čvora do 32 stepenice. Pri dohvaćanju određenog čvora moramo biti svjesni da dohvaćamo sve podatke s čvora iznad svoga te će nam neki podaci biti nepotrebni[5].

Slika 4. Prikaz umetnutog podatka u Firebase RealTime Database

Ilustracija (Slika 4.) prikazuje umetnuti redak u NoSQL bazu podataka. Iz slike se može vidjeti da je čvor “presences” podčvor glavnog korijenskog (engl. root) čvora. Isto tako uočavamo da automatski generirani ključ sadrži vrijednost drugog automatski generiranog ključa radi lakšeg povezivanja vrijednosti iz drugog čvora. Ovaj princip uzet je jer smanjuje redundantnost podatka. Također zadnji čvor prikazuje vremena u obliku unix formata.

3.2. CRUD operacije u Firebase RealTime Database

Za upisivanje podatka u bazu potrebno je instalirati biblioteku za rad s Firebase RealTime database.

```
compile: 'com.firebase:firebase-client-android:2.5.2'
compile: 'com.google.firebase:firebase-database:11.8.0'
```

Kôd 2. Prikaz biblioteke za komunikaciju s Firebase bazom

```
Firebase reference = new Firebase(Utils.FIRE_BASE_COLLEGE_REFERENCE).push();

reference.child("collegeId").setValue(reference.getKey());
reference.child("name").setValue(request.name);
reference.child("active").setValue(true);
```

Slika 5. Umetanje novog sadržaja u bazu

Uz pomoć instance Firebasea (Slika 5.) spajamo se u ovom slučaju na čvor “college”, koji je podčvor glavnog korijenskog (engl. *root*) čvora. Pomoću metode *.push(),* unutar čvora “college“ kreiramo podčvor čiji će ključ biti dužine 20 karaktera, a svaki karakter od 20 mogućih bit će unikatan. Unutar unikatnog ključa dodajemo vrijednosti “collegeId”, “name” i ”active” koji sačinavaju objekt zadanog ključa.

```
Firebase ref = new Firebase( url: Utils.FIRE_BASE_COLLEGE_REFERENCE + request.collegeId);

Map<String, Object> delete = new HashMap<>();
delete.put("active", false);

ref.updateChildren(delete);
```

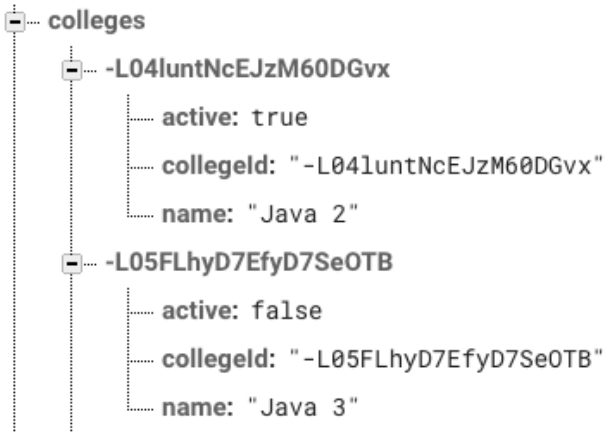
Slika 6. Uređivanje postojećeg sadržaja u bazi

Prilikom spajanja na instancu Firebasea (Slika 6.) u glavnom čvoru (engl. *root*) dohvaćamo podčvor ”college”. Unutar čvora “college” tražimo ključ koji je identičan ključu koji je poslao zahtjevatelj (engl. *request*). Nakon pronalaska određenog ključa kreiramo objekt tipa ključ-vrijednost par (engl. *key-value pair*) i u njega stavljamo vrijednost. Firebase s metodom *.updateChildren()* pronađe unutar određenog ključa objekt koji se zove “active“ te njega zamjeni s objektom koji je proslijeđen u metodu *.updateChildren()*. Trenutni isječak kôda prikazuje logičko brisanje u svojstvu uređivanja vrijednosti unutar čvora.

```
Firebase ref = new Firebase( url: Utils.FIRE_BASE_COLLEGE_REFERENCE + request.collegeId);
ref.removeValue();
```

Slika 7. Brisanje postojećeg sadržaja u bazi

Spajanjem na instancu Firebasea(Slika 7.) dohvaćamo korijen (engl. *root*), a zatim čvor “college”. Zahtjevatelj odašilje ključ te unutar čvora tražimo ključ koji je identičan zahtjevateljevom. Pozivom metode *.removeValue()* unutar čvora “college” brišemo određeni ključ iz NoSQL baze podataka.



Slika 8. Prikaz sadržaja “colleges” čvora

Nakon što je korisnik odradio operacije umetanja i uređivanja u prethodnim koracima njegov čvor (Slika 8.) spreman je za daljnje korištenje u aplikaciji.

3.3. Korišenje Firebase Storage za spremanje slika

Firebase skladište (engl. *storage*) služi za unos raznih sadržaja kao što su slike, dokumenti, audio i video zapisi. Ukoliko korisnik prilikom unošenja sadržaja u skladište (engl. *storage*) izgubi povezanost s internetom, prekinut će se unos sadržaja, no prilikom prvog spajanja na internet prijenos sadržaja će se nastaviti tamo gdje je stao. Prilikom

unošenja podataka u skladište administrator određuje korisnička prava u pogledu čitanja podataka ili i čitanja i pisanja ili uopće nema prava na pregled sadržaja. U svrhu sprečavanja "curenja informacija" administrator određuje u koje mape korisnik smije unositi svoj sadržaj. Prilikom kreiranja mape unutar skladišta treba izbjegavati znakove kao na primjer točka, ljestve, dolar (., #, $) jer to može utjecati na problem povezivanja s Firebase bazom podataka[4].

Prilikom unošenja sadržaja u skladište mogu nam se javiti slijedeće pogreške[6].

- Podatak nije pronađen (engl. *object not found*) – traženje nepostojećeg podatka unutar kante (engl. *bucket*) ili skladišta.
- Kanta nije pronađena (engl. *bucket not found*) – neuspješno spajanje na instancu gdje se nalaze svi podaci
- Potrebna autentifikacija (engl. *unauthenticed*) – korisnik se mora prijaviti da bi vidio sadržaj
- Potrebna autorizacija (engl. *unauthorized*) – korisnik nema ovlasti za pristapanju određenom sadržaju
- Otkazivanje(engl. *canceled*) – korisnik je prekinuo unošenje novog sadržaja
- Maksimalan broj pokušaja dostignut (engl. *maximum time limit*) – korisnik je dostigao maksimaln broj pokušaja u unosu, preuzimanju i brisanju sadržaja

```
compile 'com.google.firebase:firebase-storage:11.8.0'
```

Kôd 3. Prikaz biblioteke za komunikaciju s Firebase skladištem (engl. *storage*)

```
private void addUserPhoto(Uri imageUri){

    StorageReference userImageRef = storage.child("UserImages").child(uid)
            .child(imageUri.getLastPathSegment());
    userImageRef.putFile(imageUri).addOnCompleteListener(
            new OnCompleteListener<UploadTask.TaskSnapshot>() {
        @Override
        public void onComplete(@NonNull Task<UploadTask.TaskSnapshot> task) {
            @SuppressWarnings("VisibleForTests")
            Uri uploadedImageUri = task.getResult().getDownloadUrl();

            userRef.child("image").setValue(uploadedImageUri.toString());

            Toast.makeText( context: ProfileActivity.this,  text: "Image was saved
        }
    });
}
```

Slika 9. Prikaz umetanja fotografije u Firebase skladiše (engl. *storage*)

Prilikom unošenja fotografije u Firebase skladište korisnik poziva ugrađeni intent s kojim pristupa galeriji. Intenti su ugrađene poruke unutar Android operacijskog sustava s kojim se pristupa drugoj funkcionalnosti kao u ovom slučaju galeriji slika. Pri startanju intenta šalje se još i zahtjevateljev kod (engl. *request code*) s kojim se provjerava rezultat odabranog intenta unutar aktivnosti. Zahtjevateljev kod mora biti identičan kodu kojeg

primamo unutar aktivnosti. Odabrana slika sprema se u “Uri“ klasu koja sadrži podatke o slici, te se ta slika šalje u metodu *addUserPhoto()*. Unutar metode (Slika 9.) pristupa se skladištu, te se unutar njega pravi hijerarhija za svakog korisnika pri umetanju slike. Unutar metode *.putFile()* šalje se putanja slike s mobilnog uređaja. Pri završetku unošenja slike u skladište poziva se metoda *.addOnCompleteListener()* koja obavještava da je slika uspješno spremljena. Spremljena slika se nalazi unutar varijable “uploadedImageUri“ i iz tog se dobiva putanja sa servisa Firebase. Putanja iz varijable se dodijeljuje određenom korisniku i obavještava ga da je uspješno unio sliku u sustav.

3.4. Korišenje Firebase Authentication

Uz pomoć Firebase autentifikacije (engl. *authentication*) može se vrlo jednostavo kreirati korisnički račun, a isto tako i prijaviti se s kreiranim korisničkim računom. Jedna od najvažnijih značajki Firebase autentifikacije, iz perspektive korisnika, svakako je sigurnost. Firebase autentifikacija koristi Oauth 2.0 i OpenID standard za prijavu i registraciju korisnika[4]. Unutar Firebase autentifikacije postoji izbor društvenih mreža preko kojih se korisnici mogu prijaviti - Facebook, Twitter, Gmail, Github. Ako korisnik ne posjeduje niti jedan račun s društvenih mreža uvijek se može prijaviti putem svoje e-pošte ili putem mobilnog uređaja. Sagledavajući činjenice Firebase autentifikacija je najbolji izbor za prijavu i registraciju korisnika. U slučaju da korisnički podaci i dođu u ruke neovlaštene osobe, vidljiva je e-pošta ili način prijave jer lozinka korisnika se čuva na serverima kojima samo Google ima pristup[7].

```
compile 'com.google.firebase:firebase-auth:11.8.0'
```

Kôd 4. Prikaz biblioteke za komunikaciju s Firebase autentifikacijom (engl. *authentication*)

```
auth.createUserWithEmailAndPassword(request.student.getEmail(), randomPassword)
        .addOnCompleteListener(new OnCompleteListener<AuthResult>() {
    @Override
    public void onComplete(@NonNull Task<AuthResult> task) {
        if(!task.isSuccessful()){
            request.progressDialog.dismiss();
            Toast.makeText(application.getApplicationContext(), task.getException
        }else{
            auth.sendPasswordResetEmail(request.student.getEmail())
                    .addOnCompleteListener(new OnCompleteListener<Void>() {
                @Override
                public void onComplete(@NonNull Task<Void> task) {
                    if(!task.isSuccessful()){
                        request.progressDialog.dismiss();
                        Toast.makeText(application.getApplicationContext(), task.
                    }else {
```

Slika 10. Registracija studenta u Firebase

Prilikom registracije novog korisnika prikupljaju se njegovi podaci koji uključuju e-poštu, ime, prezime i spol. Pomoću softverskog paketa (engl. *SDK*) Firebase (Slika 10.) je kreirao vrlo jednostavnu metodu *.createUserWithEmailAndPassword()* u koju se prosljedi e-pošta i lozinka prikupljena s forme. Kako bi korisnik bio siguran da mu ni razvojni programer ne može otkriti lozinku prilikom kreiranja računa stvori mu se nasumično izgenerirana lozinka. Pomoću metode *.sendPasswordResetEmail()* korisniku se šalju upute za promjenu unesene lozinke. Prilikom promjene lozinke korisnik može biti siguran da je njegova lozinka strogo povjerljiva.

```
auth.signInWithEmailAndPassword(request.userEmail, request.userPassword)
        .addOnCompleteListener(new OnCompleteListener<AuthResult>() {
    @Override
    public void onComplete(@NonNull Task<AuthResult> task) {
        if(!task.isSuccessful()){
            request.progressDialog.dismiss();
            Toast.makeText(application.getApplicationContext(), task.getEx
        }else {
```

Slika 11. Prijava korisnika u aplikaciju

Korisnik unutar forme za prijavu unosi svoju e-poštu i lozinku te klikom na dugme "prijava" šalje svoje podatke u metodu (Slika 11.) *.signInWithEmailAndPassword()*. Nakon uspješne obrade zahtjeva mogu se dobiti dvije povratne informacije. Prva informacija je neuspješna prijava u sustav, te se korisnika automatski obavještava pomoću iskočnog prozora. Druga informacija je uspješna prijava u sustav i korisnika se prebacuje na početnu stranicu.

4. Uvod u MySQL bazu podataka

Baze podatka[8] temeljno su središte svake aplikacije. Podatke unutar baze možemo zapisati, urediti, izbrisati i dohvatiti podatke koje zahtjeva korisnik. Svaka relacijska[9] baza podataka sastoji se od tablica, a svaka tablica unutar sebe sadrži retke i stupce. Tablica unutar sebe sadrži primarni ključ koji određuje cijeli redak kao unikatnu stavku u cijelom skupu redaka. Neke tablice unutar sebe mogu sadržavati i strani ključ koji služi za smanjenje redundantnosti između podataka. U bazama podataka teži se da je svaki podatak samo jednom zapisan u tablicu radi lakše izmjene u budućnosti.

MySQL baza podataka je najomiljenija baza podataka u svijetu PHP programera. Jedan od razloga tome leži u činjenici da unutar softverskog paketa "LAMP" (Linux, Apache, MySQL, PHP)[10] dolazi već ugrađena MySQL baza podataka. Baza MySQL je server koji osluškuje upite klijenta te klijentu natrag vraća rezultat. Klijent je svaki spojeni korisnik koji ima dopuštenje slanja upita na server. Kako bi klijent uspješno mogao poslati upit na bazu, aplikacija mora biti konfigurirana s odgovarajućim pristupnim podacima. Pristupni podaci za MySQL bazu sadržavaju ime poslužitelja, ime baze, korisničko ime i lozinku. U slučaju da se MySQL baza nalazi na poslužitelju gdje je instaliran i Apache web server, ime poslužitelja bit će lokalni poslužitelj (engl. *localhost*) što označava lokalnu adresu Apache web servera. Kreiranje baze započinje čarobnjakom MySQL (engl. *database wizard*) gdje se unosi ime nove baze te naposljetku korisničko ime i lozinka za tu bazu.

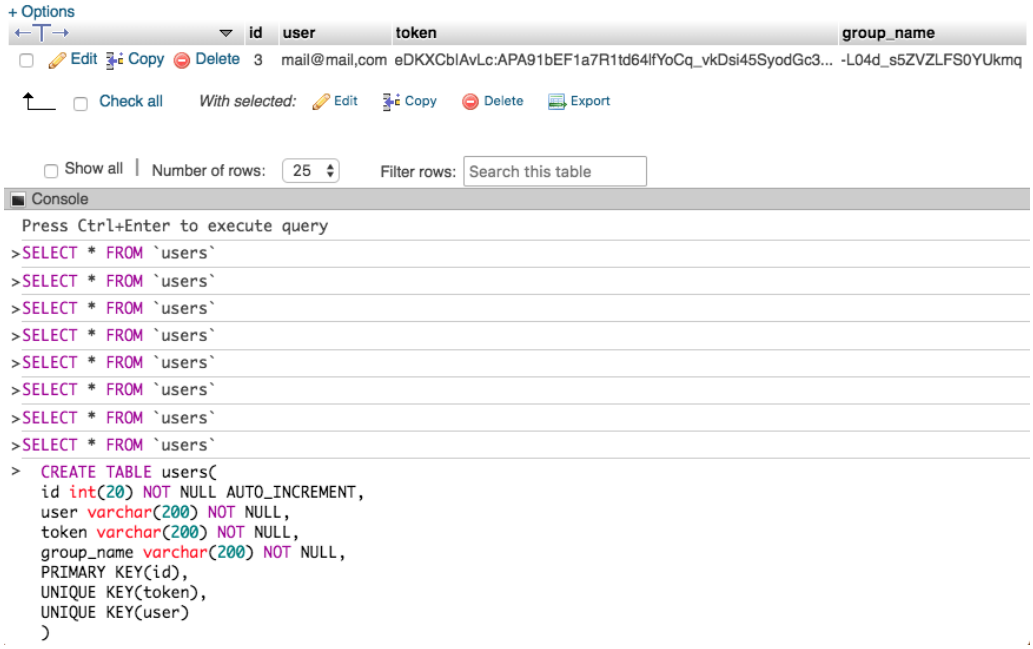
Za administraciju MySQL baze podataka obično se koristi "phpMyAdmin". Softver phpMyAdmin otvorenog je kôda (engl. *open source*) i napisan je u PHP programskom jeziku. PhpMyAdmin[11] je grafički alat za upravljanje tablicama, pokretanja SQL upita, praćenja izmjena nad bazom, mjenjanje korisničkih rankova, prikazivanje podataka u različitim formatima, kao naprimjer CSV, PDF, XML i još mnogo toga.

4.1. Korišenje MySQL baze podataka za spremanje tokena

```
CREATE TABLE users(
    id int(20) NOT NULL AUTO_INCREMENT,
    user varchar(200) NOT NULL,
    token varchar(200) NOT NULL,
    group_name varchar(200) NOT NULL,
    PRIMARY KEY(id),
    UNIQUE KEY(token),
    UNIQUE KEY(user)
)
```

Kôd 5. Prikaz SQL skripte za kreiranje nove tablice

Pomoću naredbe kreiranje tablice (Kod 5.) (engl. *create table*) kreiramo korisnika koji sadrži tipove podataka, kao na primjer cijeli broj (engl. *int*) i tekstualni zapis (engl. *varchar*). Broj unutar okruglih zagrada predstavlja koliko karaktera može maksimalno biti unutar jedne ćelije. Naredba zabrana praznome (engl. *not null*) govori da se ne smije spremiti prazan unos u ćeliju. Prilikom svakog novog unosa korisnika u bazu naredba automatsko povećanje (eng. *auto increment*) rješava problem iste identifikacijske oznake u bazi. Naredba primarni ključ (engl. *primary key*) postavlja se na onu kolonu za koju se zna da će uvijek biti unikatna. Također naredba primarni ključ mogla se još staviti i na kolonu token i kolonu korisnik (engl. *user*). Uz pomoć naredbe unikatni ključ (engl. *unique key*) dodjeljujemo svojstva određenoj koloni da ne smije imati dvije iste vrijednosti.



Slika 12. Prikaz kreiranja tablice u phpMyAdmin

Uz pomoć ilustracije (Slika 12.) može se vidjeti kompletno sučelje phpMyAdmina. Posebno zanimljiva stavka je konzola (engl. *console*) unutar koje unosimo naredbe nad novo kreiranom bazom ili tablicom. Unutar konzole može se kreirati novu tablicu(Kôd 5.) ili izvršiti upit nad već postojećom tablicom. Također na ilustraciji je vidljiva i tablica s jednim unesenim korisnikom. Kod korisnika se može vidjeti da mu identifikacijska oznaka nije jedan već tri, to je iz razloga što su prijašnji korisnici izbrisani. Identifikacijska oznaka ne smije imati broj koji je prije već postojao u tablici iako se taj broj više ne koristi.

5. Uvod u PHP programski jezik

Programski jezik PHP jedan je od najpopularnijih skriptnih jezika koji se koristi u današnje vrijeme. Više od 60 % web stranica koristi PHP jezik. Za toliki postotak najviše su zaslužne platforme Wordpress, Magento, Drupal, Joomla i ostali. PHP programski jezik započeo je s proceduralnim stilom pisanja koda, ali zbog teškog održavanja kôda verzije 3 uveden je i objektno orijentiran stil pisanja kôda. Današnja zadnja verzija PHP-a je verzija 7.2. Najzastupljenija verzija PHP-a je verzija 5. Skriptni jezik PHP još uvjek je zadržao opciju pisanja kôda u proceduralnom stilu[12].

Ukoliko se razvija projekt u kojem će biti mnogo izmjena i nadogradnje objektno orijentirani stil je najbolji pristup za održavanje koda. Uz pomoć objektno orijentiranog stila rješavaju se problemi kao na primjer ponovno iskorištavanje postojećih funkcionalnosti, nadogradnja postojećih funkcionalnosti i održavanje postojećih funkcionalnosti. Ukoliko se projekt piše proceduralnim stilom, mora se imati na umu da će se mnogo funkcionalnosti morati prepisivati što dovodi do loše arhitekture samog projekta. Nakon nekog vremena uslijed mnogobrojnih izmjena kôda projekt će se morati prepisati na objektno orijentirani stil radi lakšeg korištenja[12].

```
class Push {
    private $title;
    private $message;

    function __construct($title, $message) {
        $this->title = $title;
        $this->message = $message;
    }

    public function getPush() {
        $res = array();
        $res['data']['title'] = $this->title;
        $res['data']['message'] = $this->message;
        return $res;
    }

}
```

Slika 13. Klasa u duhu objektno orijentiranog stila

Objektno orijentirani stil započinje kreiranjem klase (engl. *class*) (Slika 5.). Svaka klasa sadrži svojstva (engl. *property*), u ovom slučaju naslov i poruka. Unutar funkcije "__construct" unosimo naslov i poruku te time kreiramo objekt tipa push.

```
$push = new Push("Main title", "My message");
$notification = $push->getPush();
```

Kôd 6. Prikaz kreiranja i dohvaćanja novog objekta

Kreirani objekt spremi se u varijablu (Kod 6.) te se zatim nad tom vrijablom poziva metoda *getPush()* s kojom dohvaćamo podatke iz inicijaliziranog objekta.

5.1. Korištenje web servisa u PHP programskom jeziku

Web servis[13] je aplikacija koja je smještena na serveru i podržava interakciju s korisnikom putem http zaglavlja. Unutar http zaglavlja postoje četiri metode koje se koriste za komunikaciju i one su prikazane u tablici (Tablica 2.).

Tablica 2. Prikaz http metoda

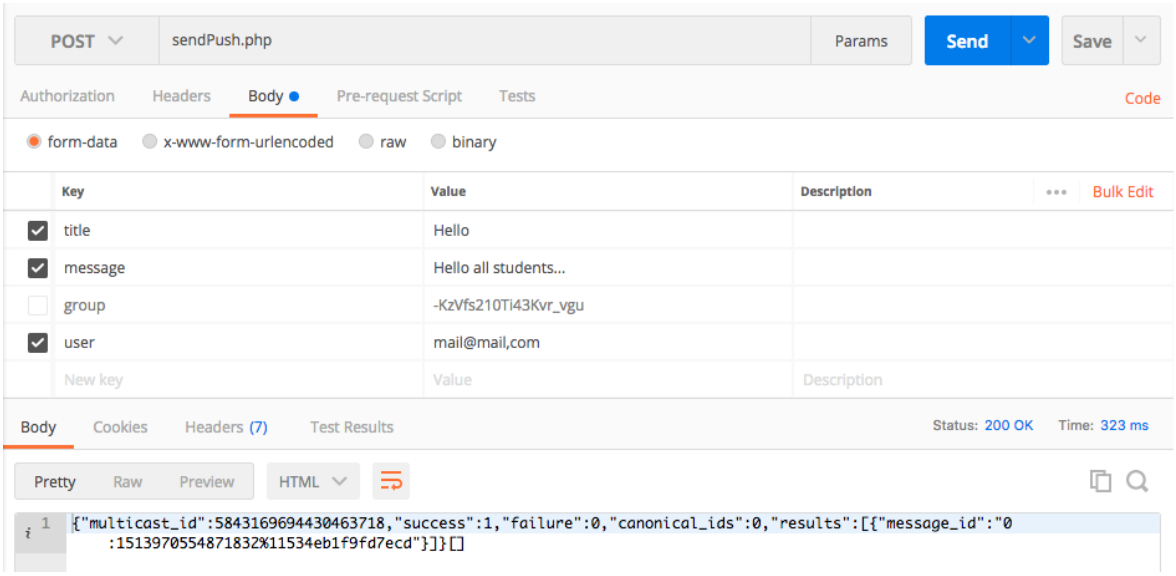
GET	Služi za dohvat podataka
PUT	Služi za ažuriranje podataka
POST	Služi za unos novog podatka
DELETE	Služi za brisanje podatka

Ove četiri metode su ekvivalentne metodama unutar jezika SQL

- Metoda GET je ekvivalentna naredi SELECT unutar SQL jezika
- Metoda PUT je ekvivalentna naredi UPDATE unutar SQL jezika
- Metoda POST je ekvivalentna naredi INSERT unutar SQL jezika
- Metoda DELETE je ekvivalentna naredi DELETE unutar SQL jezika

Prilikom dohvaćanja podataka mogu se dobiti slijedeći http kôdovi[25]

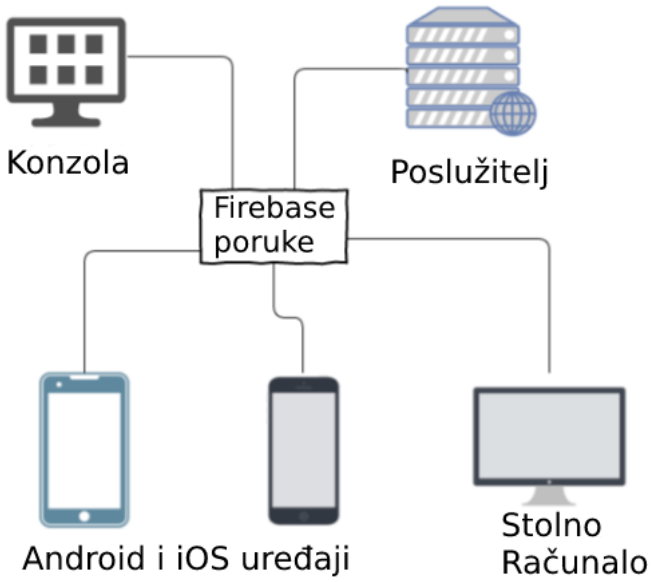
- Http kôd 200 obavještava da su podaci uspješno dohvaćeni
- Http kôd 401 obavještava da niste prijavljeni za pregled podataka, odnosno da korisnik nije autentificiran
- Http kôd 403 obavještava da nemate dovoljna prava za pregled podataka, odnosno da korisnik nije autoriziran za pregled
- Http kôd 404 obavještava da određeni servis ne postoji
- Http kôd 500 obavještava da se desila određena greška na serveru



Slika 14. Prikaz http poziva prema web servisu

Ilustracija (Slika 14.) prikazuje http pošalji (engl. *post*) zahtjev prema poslužitelju. Prilikom pošalji (engl. *post*) zahtjeva parametri se šalju u tijelo (engl. *body*) zahtjeva. Kada je zahtjev poslan prema poslužitelju na dnu ekrana može se vidjeti http status kôd 200 koji obavještava da je uspješno zahtjev zaprimljen. Ako je sve uspješno prošlo kao u ovom primjeru na dnu ekrana može se vidjeti JSON[14]. U ovom primjeru JSON prikazuje uspješno slanje obavijesti na mobilni uređaj. Ovaj http post zahtjev napravljen je unutar aplikacije Postman[15]. Postman se koristi za testiranje API-ja te je to najbrži i najlakši način za provjeru rada web servisa.

5.2. Korištenje Firebase Cloud Messaging s PHP programskim jezikom



Slika 15. Prikaz rada Firebase platforme za poruke

Platforma Firebase[4] za razmjenu poruka (Slika 15.) u oblaku (engl. *cloud messaging*) je platformsko rješenje za slanje obavijesti u stvarnom vremenu na više različitih uređaja. Za korištenje platforme potrebno je imati svoj vlastiti poslužitelj (engl. *server*) koji šalje preko posta zahtjeva poruke na mobilni i web uređaj. Ukoliko nemamo vlastiti server postoji opcija slanja poruke direktno putem Firebase konzole, ali tu postoje ograničenja. Svaki korisnik unutar web ili mobilne aplikacije može se prijaviti na određenu temu (engl. *topic*) te dobivati obavijesti isključivo vezane za sadržaj koji je odabrao. Primjerice ako se web stranica ili mobilna aplikacija bavi sportskim sadržajem korisnik se može pretplatiti na temu nogomet i dobivat će informacije samo za taj sport. Ovim rješenjem korisnik dobiva sadržaj koji mu je interesantan.

Ako se koristi Firebase konzola za slanje poruka postoji ograničenje da samo administrator može slati obavijesti te veličina poruke ne smije prelaziti 2 kilobajta. Unutar konzole može se automatizirati vrijeme slanja poruke tako da korisnik ima osjećaj da je administrator stranice uvijek prisutan. Pri izboru slanja poruke određena poruka može se poslati samo jednom korisniku ili više korisnika slanjem na određenu temu. Za slanje poruke određenom korisniku potreban je token koji se dobiva autentifikacijom korisnika u sustav.

Slanjem poruke na određenu temu nije potreban token jer je korisnik automatski postavio svoj token pretplatom na temu.

Prilikom korištenja vlastitog servera mogućnosti su puno veće kao na primjer korisnik može korisniku poslati poruku bez administratora. Sva komunikacija ide preko vlastitog servera tako da administrator više ne mora biti posrednik u slanju poruka. Veličina poruke može varirati od 2 kilobajta do 4 kilobajta jer ovisi o tome kakav JSON se šalje prema serveru. U slučaju da se unutar JSON objekt nalazi podobjekt obavijest (engl. *notification*) maksimalna veličina poruke smije biti 2 kilobajta. Ako se unutar JSON nalazi podobjekat podatak (engl. *data*) maksimalna veličina poruke smije biti 4 kilobajta. Pri prikazivanju obavijesti na mobilnom uređaju ista metoda obrađuje oba poslana podobjekta**Pogreška! Knjižna oznaka nije definirana.**.

Prilikom pokretanja aplikacije u pozadini se podigne servis koji osluškuje nove poruke. Svaka poruka koja pristigne na mobilni uređaj može biti vidljiva unutar aplikacije (engl. *foreground*) i van aplikacije (engl. *background)* kad korisnik ne gleda aplikaciju.

```php
$url = 'https://fcm.googleapis.com/fcm/send';

$headers = array(
    'Authorization: key=' . FIREBASE_API_KEY,
    'Content-Type: application/json'
);

$ch = curl_init();

curl_setopt($ch, CURLOPT_URL, $url);
curl_setopt($ch, CURLOPT_POST, true);
curl_setopt($ch, CURLOPT_HTTPHEADER, $headers);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, false);
curl_setopt($ch, CURLOPT_POSTFIELDS, json_encode($fields));

$result = curl_exec($ch);
if ($result === FALSE) {
    die('Curl failed: ' . curl_error($ch));
}

curl_close($ch);
```

Slika 16. prikaz slanja poruke na FCM

Ova ilustracija (Slika 16.) koja prikazuje način komunikacije preko cURL-a. cURL ovdje se koristi kako bi se podaci unutar JSON-a poslali direktno na usklađeni lokator sadržaja (engl. *url*).

```
curl -X POST --header "Authorization: key=FIREBASE_API_KEY" \
    --Header "Content-Type: application/json" \
    https://fcm.googleapis.com/fcm/send \
    -d "$fields"
```

Kôd 7. Prikaz slanja post zahtjeva putem cUrl-a[16]

Ovim prikazom(Kôd 7.) vidljivo je kako izgleda poslani zahtjev(Slika 16.) sa ciljem dobivanja obavijesti na mobilnom uređaju. Može se primijetiti da se u zaglavlju šalje ključ koji je izdala platforma Firebase i tip podatka koji se šalje prema serveru je JSON. Oznaka "-d" označava podatak ili tijelo (engl. *body*) koje se šalje na određeni usklađeni lokator sadržaja[26] (engl. *url*).

6. Funkcionalnosti Android aplikacije na mobilnom uređaju

Aplikacija na mobilnom uređaju sastoji se od tri role, a to su administrator, profesor i student. Prilikom pokretanja aplikacije prvi kreirani korisnički račun je administrator. Administrator unutar aplikacije postavlja sve postavke vezane uz raspored, prijavu i registraciju novih korisnika, uređivanje postojećih korisnika, određivanje broja predavanja i vježbi. Profesor unutar aplikacije dnevno prati studente, te ih obavještava o nekom događaju. Student unutar aplikacije prati svoj trenutni raspored, postavlja si bilješke, te prati dolaznost za svaki kolegij. Svaka rola može urediti svoj vlastiti profil postavljanjem slike.

6.1. Funkcionalnost role administrator

Rola administrator unutar aplikacije može kreirati grupu ili je izbrisati, a u slučaju pogreške urediti. Administratorska rola istu ulogu ima i kod kolegija (kreiranje, brisanje ili uređivanje). Administrator unutar sustava može kreirati učionice, urediti imena učionica ili ih izbrisati, te u rasporedu dodijeliti određenoj grupi. Unutar aplikacije administrator može kreirati korisničke račune za role profesor, student i time onemogućiti vanjski utjecaj unutar aplikacije.

Prilikom dodavanja studenta u sustav, studentu se dodjeljuje ime, prezime, e-pošta, jmbag, spol, te grupa. Ako je registracija studenta uspješna, studentu na adresu e-pošte pristiže obavijest da si postavi novu lozinku s kojom se prijavljuje u sustav. Za uspješno praćenje studenta postoji rola profesor. Rola profesor prilikom registracije mora sadržavati ime, prezime, e-poštu, titulu, spol, te popis kolegija i njegov tip kao na primjer predavanja ili vježbe. Prilikom uspješne registracije profesoru pristiže obavijesti na e-poštu gdje si postavlja lozinku s kojom se prijavljuje u sustav. Obje role administrator može ažurirati kao na primjer promijeniti određenom studentu grupu, a da pritom ne utječe na dosadašnje dolaznosti. Studentu i profesoru se mogu urediti svi parametri koju su bili zahtijevani kod registracije. Prilikom uređivanja role profesor, administrator može izmijeniti kolegije i tipove koje vodi dotični profesor, te on vidi novi raspored samo za te kolegije.

Za svaku grupu administrator kreira raspored. Prilikom kreiranja rasporeda odabire se dan i postavlja se početno i završno vrijeme za svaki kolegij. Izborom kolegija bira se tip

kolegija kao na primjer jesu li to vježbe ili predavanja, te posljednja stavka je izbor učionice. U slučaju da administrator nije zadovoljan trenutnim rasporedom može ga lako izbrisati. Pri kreiranju rasporeda administrator ne mora paziti vremensko preklapanje kod određene grupe već ga u tome upozorava sama aplikacija. Administrator također postavlja granicu u sustavu koliko je dolaznosti potrebno za vježbe, a koliko za predavanja.

6.2. Funkcionalnost role profesor

Profesor unutar sustava može urediti svoj vlastiti profil postavljanjem slike ili promjenom svog vlastitog imena i prezimena. Rola profesor može vidjeti koje sve kolegije vodi, te odabirom kolegija odabire i grupu.

Prilikom odabira grupe profesor može vidjeti datum i vrijeme održavanja, te završetak kolegija, a zatim i učionicu u kojoj se izvode predavanja. Odabirom datuma po želji profesor može vidjeti sve studente koji trebaju prisustvovati, bez obzira na grupu. U slučaju da postoje studenti koji nisu iz grupe koja treba prema rasporedu slušati predavanja, pored tih studenata stoji znak "(!)". Uz pomoć znaka "(!)" profesor može zaključiti da ti studenti nisu obavezni biti na trenutnim predavanjima. Svaki student koji prođe pored uređaja za prepoznavanje lica, automatski se profesoru na ekranu pored dotičnog imena studenta pojavi kvačica da je prisutan na današnjim predavanjima.

Putem obavijesti profesor može obavijestiti svaku grupu o važnom događaju. Profesor bira kolegij, piše određenu poruku za grupu i šalje studentima. Student kada primi poruku može točno vidjeti od kojeg profesora je pristigla poruka.

6.3. Funkcionalnost role student

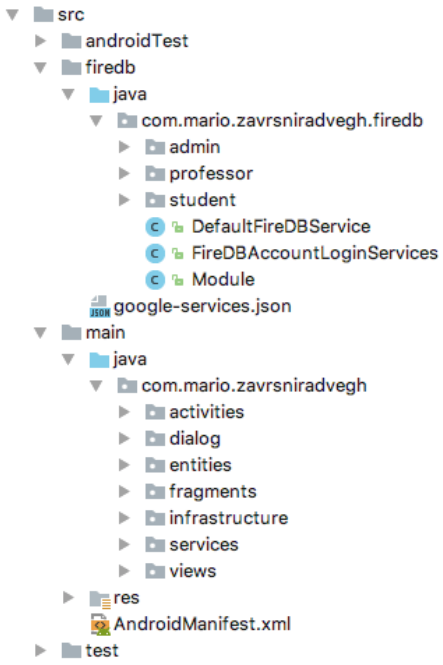
Student unutar sustava može urediti vlastiti profil postavljanjem svoje slike. Klikom na raspored student može pogledati za svaki dan trenutni raspored. U slučaju da student nije došao na određeno predavanje unutar svoje grupe, može vidjeti predavanja od druge grupe i prisustvovati im u koliko je slobodan.

Predavanja označena sa znakom "(!)", govore studentu, da ta predavanja nisu obavezna za njegovu grupu, nego samo za njega ako želi skupiti što više dolaznosti kako bi zadovoljio kriterije za uspješnu dolaznost. Odabirom kolegija iz rasporeda student može vidjeti svoju prisutnost na predavanjima. Također student može vidjeti koliko dolaznosti mu

nedostaje da bi uspješno zadovoljio dolaske na određeni kolegij. Da bi student uspješno zadovoljio dolaske na kolegij mora imati pola ili više dolaznosti od maksimalnog broja koje je postavio administrator.

Svaki dan jedan sat prije početka predavanja studentu stiže obavijest na mobilni uređaj da mora krenuti prema fakultetu. Student pored svakog kolegija ima notez u koji može zapisivati važne bilješke za kolegij. Također student može za svaki kolegij urediti notez ili ga obrisati, te ga ponovo kreirati.

7. Struktura Android aplikacije na mobilnom uređaju



Slika 17. Prikaz Android strukture na mobilnom uređaju

Ilustracija (Slika 17.) prikazuje izgled strukture Android aplikacije. Početni projekt se nalazi u glavnoj (engl. *main*) mapi. Glavna mapa se sastoji od podmape java te zatim slijedi ime paketa (engl. *package name*). Unutar paketa nalazi se sedam podpaketa koji čine Android aplikaciju.

Unutar paketa aktivnosti (engl.*activity*) nalaze se podpaketi administrator, student i profesor. Svaki podpaket čini jednu rolu i prikazuje aktivnost vezanu samo za tu rolu. Aktivnosti koje su zajedničke za sva tri ranka nalaze se samo unutar paketa aktivnost. Aktivnost je ekran koji prikazuje korisničko sučelje, na primjer registracija studenta. Dijalog (engl. *dialog*) paket je podijeljen na tri podpaketa koje čine administrator, student i profesor. Dijalozi su iskočni prozori u koje korisnik unosi tekst, kao na primjer dodavanje novog kolegija. Paket entitet (engl. *entities*) prikazuje klase unutar aplikacije, na primjer klasa kolegij. Unutar paketa fragment se nalaze tri podpaketa administrator, profesor i student. Fragmenti su manje cjeline unutar jedne aktivnosti, a svaka manja cjelina se sastoji od svoje logike i sučelja, na primjer prikaz rasporeda studentu za svaki dan. Unutar paketa infrastruktura (engl. *infrastructure*) nalazi se početna klasa koja se pokreće pri otvaranju

aplikacije. Unutar početne klase radi se inicijalizacija Firebase baze podataka te inicijalizacija autobusa s događajima (engl. *event bus*) koja služi za odvajanje funkcionalnosti u više različitih paketa ili klasa. Unutar paketa servis (engl. *service*) nalaze se tri podpaketa administrator, student i profesor te podpaket vezan za obavijesti od Firebase. Unutar tri paketa vezana za korisnike su klase koje služe za razmjenu poruka između odvojenih komponenti putem autobusa s događajima (engl. *event bus*). Unutar paketa pogledi (engl. *views*) također su tri podpaketa vezana za svaku rolu u sustavu. Za svaku rolu su prikazani pogledi, kao na primjer ispis liste svih kolegija kojima treba prisustvovati student.

Paket baza Firebase skraćeno "firedb" prikazuje komunikaciju s bazom podataka Firebase putem autobusa s događajima (engl. *event bus*). Unutar svakog paketa za svaku rolu nalazi se odgovarajuća logika s funkcijama koje može obaviti. Unutar glavnog paketa "firedb" nalazi se datoteka google-servis (engl. *service*) koja je registrirana samo za paket "firedb" tako da sva komunikacija mora proći kroz "firedb" paket prije nego što podatak pristigne u bazu.

7.1. Korištenje autobusa s događajima

Autobus s događajima[17] (engl. *event bus*) koristi se kako bi se smanjila kompleksnost kôda. Uz pomoć autobusa s događajima rješavaju se problemi:

- Povezanost velike količine kôda
- Smanjenje ponavljajućih dijelova kôda
- Lakše praćenje toka izvedbe kôda
- Smanjenje količine grešaka u kôdu

Autobus s događajima predstavlja arhitekturu pošiljatelj - (engl. *publisher*) pretplatnik (engl. *subscriber*). Unutar početne klase (engl. *class*) potrebno je napraviti registraciju autobusa s događajima (engl. *event bus*) koja služi kao početna točka za slanje poruka. Sve ostale komponente u aplikaciji koje žele komunicirati putem autobusa s događajima (engl. *event bus*) moraju koristiti već postojeću instancu.

```
compile 'com.squareup:otto:1.3.8'
```

Kôd 8. Prikaz biblioteke za autobus s događajima (engl. *event bus*)

```
public static class ChangeGroupRequest{
    public String groupId;
    public String name;

    public ChangeGroupRequest(String groupId, String name) {
        this.groupId = groupId;
        this.name = name;
    }
}
 bus.post(new AdminGroupServices.ChangeGroupRequest(groupId, changeGroup.getText().toString()));
@Subscribe
public void ChangeGroup(AdminGroupServices.ChangeGroupRequest request){
    AdminGroupServices.ChangeGroupResponse response = new AdminGroupServices.ChangeGroupResponse();

    if(request.name.isEmpty()){
        response.setPropertyErrors( property: "group",  error: "Group must have a name");
    }
    if(response.didSuceed()) {
        Firebase reference = new Firebase( url: Utils.FIRE_BASE_GROUPS_REFERENCE + request.groupId);

        Map<String, Object> group = new HashMap<>();
        group.put("name", request.name);

        reference.updateChildren(group);
    }

    bus.post(response);
}
```

Slika 18. Prikaz korištenja autobusa s događajima (engl. *event bus*)

Prije korištenja autobusa s događajima (Slika 18.) (engl. *event bus*) potrebno je kreirati klasu (engl. *class*) koja će primati identičan parametar kakav prima metoda koja je pretplaćena na događaj. U ovom slučaju kreirana je klasa (engl. *class*) koja služi za uređivanje grupnog imena i prima identifikacijsku oznaku grupe te naziv grupe. Autobus s događajima (engl. *event bus*) unutar metode pošalji (engl. *post*) prima klasu za uređivanje grupe s argumentima za izmjenu grupe. Metoda koja je pretplaćena na točno određeni parametar u ovom slučaju promjeni grupu (engl. *change group*), mora iznad sebe imati anotaciju pretplata (engl. *subscribe*) koja govori da je unutar mehanizma autobusa s događajima (engl. *event bus*). Putem parametarske varijable zahtjev (engl. *request*) pristupa se argumentima koji su bili proslijeđeni u metodu pošalji (engl. *post*). Parametri unutar varijable zahtjev (engl. *request*) služe za spremanje podataka u NoSQL bazu Firebase.

8. Python Desktop aplikacija

Za korištenje grafičkog sučelja (engl. *gui*) unutar Python programskog jezika koristi se biblioteka "Tkinter" i ona već dolazi ugrađena u sam jezik. Uz pomoć Tkinter[18] biblioteke može se s vrlo malom količinom kôda složiti moćno grafičko sučelje. Grafičko sučelje koje je kreirano unutar operacijskog sustava "Windows" identično se prikazuje i na operacijskom sustavu "Linux" što znači da je biblioteka podržana u više operacijskih sustava. Unutar operacijskog sustava "Mac OS X" grafičko sučelje će biti približno identično ali nikad ne isto kao unutar operacijskog sustava "Linux" i "Windows".

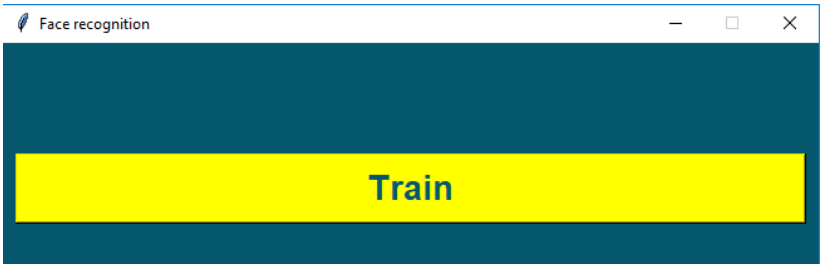
Biblioteka "Tkinter" unutar sebe ima više od petnaest ugrađenih komponenti. Unutar ovog rada korištene su komponente dugme, tekstualni zapis, prozor s porukom te iskočni prozor za unos poruke.

```
import tkinter as tk
from tkinter import font
root = tk.Tk()
my_font = font.Font(family='Helvetica', size=20,
weight='bold')
train_btn = tk.Button(root, text="Train", font="my_font",
command=train, height=1, width=15, background="#ffff000"
foreground="#05576d")
train_btn.pack(side="top", fill="x" expand="yes", padx=10,
pady=10)
root.mainloop()
root.quit()
```

Kôd 8. Prikaz kreiranja gumba u biblioteci "Tkinter"

Prilikom kreiranja aplikacije (Kôd 8.) potrebno je uključiti biblioteku "Tkinter" korištenjem naredbe uvoz (engl. *import*). Unutar varijable "root" poziva se inicijalizacija biblioteke za koju se vežu sve komponente koje će biti prikazane na ekranu. Prilikom kreiranja dugmadi potrebno se referencirati na uvoznu biblioteku te zatim pozvati metodu dugme[18] (engl. *button*). Unutar dugmadi postavlja se referenca na inicijalizacijsku varijablu, dodaje se tekst, događaj na klik te stilističko uređenje gumba. Uz pomoć metode *.pack()* pozicionira se dugme unutar grafičkog sučelja. Kako bi se sve iscrtalo na ekran i prikazalo korisniku potrebno je pozvati metodu *.mainloop()* na inicijalizacijskoj varijabli.

U slučaju da želimo aplikaciju zatvoriti potrebno je pozvati metodu *.quit()* na incijalizacijskoj varijabli.



Slika 19. Prikaz gumba unutar Python desktop aplikacije

8.1. Komunikacija s bazom Firebase

Komunikacija Python desktop aplikacije s Firebaseom[19] složena je putem REST API-ja. Za komunikaciju je korištena biblioteka "python-firebase".

```
pip install python-firebase
```

Kôd 9. Instalacija python-firebase biblioteke

```
import requests

#used python-firebase
from firebase import firebase
firebase = firebase.FirebaseApplication('https://zavrsnirad.firebaseio.com', None)
result = firebase.get('/users', 'mario,vegh@racunarstvo,hr')
print("python-firebase")
print(result)

#without python-firebase
res = requests.get("https://zavrsnirad.firebaseio.com/users/mario,vegh@racunarstvo,hr.json")
print("only requests")
print(res.content)

enshoot
/Library/Frameworks/Python.framework/Versions/3.5/bin/python3.5 /Users/mvegh/PycharmProjects/
python-firebase
{'email': 'mario.vegh@racunarstvo.hr', 'group': '-L04d_s5ZVZLFS0YUkmq', 'sex': 'Male', 'name'
only requests
b'{"email":"mario.vegh@racunarstvo.hr","group":"-L04d_s5ZVZLFS0YUkmq","image":"","jmbag":"123
```

Slika 20. Prikaz dohvaćanja podata iz Firebase putem REST API

U ovom primjeru (Slika 20.) prikazane su dvije mogućnosti za dohvaćanje podataka. Prvi primjer prikazuje dohvaćanje podataka korištenjem biblioteke "python-firebase". Drugi primjer prikazuje dohvaćanje podataka putem ugrađene biblioteke zahtjevi (engl. *requests*) u samom Pythonu. Razlog korištenja ugrađene biblioteke "python-firebase[19]" je u smanjenju kompleksnosti kôda unutar same aplikacije.

```
import requests

#used python-firebase
from firebase import firebase
firebase = firebase.FirebaseApplication('https://zavrsnirad.firebaseio.com', None)
firebase.post('/screenshot', {"Name": "Mario", "Surname": "Vegh"})

#used python-firebase
result = firebase.get('/screenshot', None)
print("python-firebase")
print(result)

#without python-firebase
res = requests.get("https://zavrsnirad.firebaseio.com/screenshot.json")
print("only requests")
print(res.content)
```

```
eenshoot
/Library/Frameworks/Python.framework/Versions/3.5/bin/python3.5 /Users/mvegh/Pychar
python-firebase
{'-L1Ne7coAkX29WCdxivq': {'Surname': 'Vegh', 'Name': 'Mario'}}
only requests
b'{"-L1Ne7coAkX29WCdxivq":{"Name":"Mario","Surname":"Vegh"}}'
```

Slika 21. Prikaz slanja podataka na Firebase putem REST API

Trenutna slika (Slika 21.) prikazuje slanja podataka u NoSQL bazu Firebase putem biblioteke “python-firebase“. Potrebno se spojiti na instancu Firebasea te poslati unutar metode *.post()* čvor u koji se želi spremiti podatak u obliku JSON-a. Čvor unutar Firebase baze podataka predstavlja ključ, dok JSON podaci predstavljaju vrijednost pod određenim ključem. Uz pomoć metode *.get()* nad firebase instancom dohvaćaju se svi podaci iz trenutnog čvora te se prikazuju u JSON obliku korisniku na ekran.

9. OpenCV biblioteka

OpenCV biblioteka je najpopularnija biblioteka otvorenog koda(engl. *open source*) i služi za razvijanje aplikacija koje se bave računalnim vidom. Inicijalno je biblioteku kreirao Intel istraživački centar 1999. godine dok je 2008. godine kompanija Willow Garage nastavila pružati podršku i razvoj. Posebnost ove biblioteke je procesiranje slika u stvarnome vremenu te je dostupna na svim operacijskim sustavima i to na: Windowsu, Linuxu i Mac OS Xu, Androidu, iOSu[20].

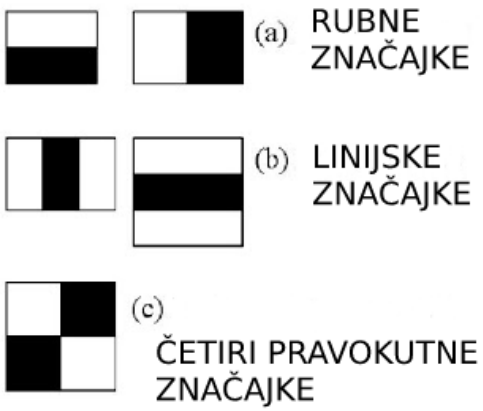
Trenutno biblioteka ima ugrađeno više od 2500 optimiziranih algoritama, a podržana je u jezicima C++, Python i Java, Matlab i C. Uz pomoć OpenCV biblioteke mogu se raditi operacije na slikama. Na primjer može se uslikati sliku uz pomoć kamere, te ju kasnije obraditi. Nakon učitavanja fotografije unutar aplikacije, fotografija je prikazana u obliku (nxn) matrice. Obrađivanje slika moguće je i uz korištenje algoritma unutar same biblioteke ili uz pomoć vlastitog algoritma. Uz pomoć biblioteka moguće je detektirati razne objekte kao na primjer oči, usta, lice, aute, registarske pločice, dio osobe ili pak cijelu osobu. Također moguće je detektirati različite oblike na slici, na primjer krug koji može prikazivati kovanice ili pravokutnik koji može prikazivati papirnate novčanice. S bilo koje slike mogu se izdvojiti slova na primjer detektira se registarska pločica. Uz pomoć algoritma za segmentaciju slike složi se oblik određenog slova. Uz pomoć bilo koje slike mogu se raditi njezine manipulacije, na primjer promjena veličine, rezanje, izoštravanje, zamućenja, rotacije, zamjena položaja dijelova slika[21].

Trenutno OpenCV broji više od 47 tisuća korisnika putem foruma te više od 14 milijuna preuzimanja. Ova biblioteka koristi se u akademskim zajednicama, vladinim tijelima, istraživačkim skupinama te velikim kompanijama poput Googlea, Yahooa, Microsofta, Intela, IBM-a[20].

```
pip install opencv-python
```

Kôd 10. Instalacija[27] OpenCV biblioteke unutar Python-a

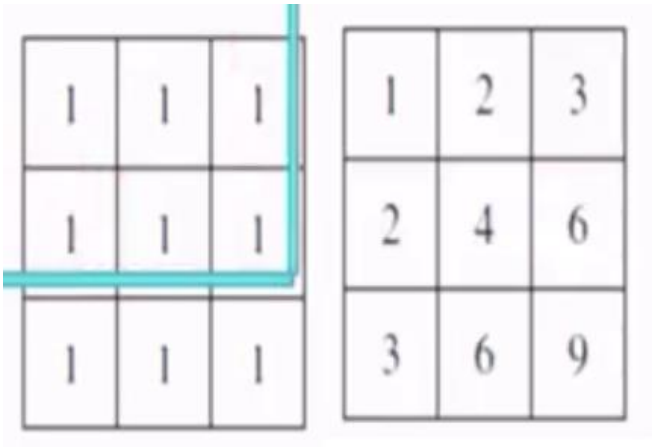
9.1. Detekcija lica putem haar kaskada



Slika 22. Prikaz haar značajke[22]

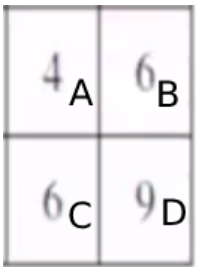
Detekcija objekata putem haar[22] značajki jedna je od najučinkovitijih metoda detekcije objekata koju su predložili Paul Viola i Michael Jones unutar svoga rada iz 2001. godine. Za korištenje metode koristi se strojno učenje bazirano na mnogo pozitivnih i negativnih slika. Pod pozitivnim slikama smatraju se slike koje unutar sebe sadrže lice, dok kod negativnih slika nema lica, već sama pozadina. Izdvajanje značajki lica prikazano je na slici(Slika 22.). Svaka dobivena značajka pojedinačna je vrijednost dobivena razlikom sume bijelog i crnog pravokutnika.

Kao početna pozicija za izdvajanje značajki uzima se dio slike koji je veličine 24x24 piksela i unutar njega ima više od 160 tisuća značajki. Ako na primjer imamo sliku od 72x72 piksela bazna pozicija za izdvajanje značajki pomaknut će se 3 puta što znači da će cijela slika imati više od 480 tisuća značajki. Na slici (Slika 22.) mogu se vidjeti crni i bijeli pravokutnici za svaku značajku. Svaki pravokutnik prikazuje 1 piksel i uvijek jedna značajka mora biti minimalno 2 piksela što znači da se mora sastojati od jednog bijelog polja i jednog crnog polja. Za dobivanje integrala slike bijeli pravokutnik može se zamijeniti s brojem -1, a crni pravokutnik s brojem 1[22].



Slika 23. Izračuna integrala slike unutar crnog polja

Integral slike koristi se kako bi se ubrzalo računanje haar značajki. Crno polje može se zamisliti kao niz jedinca koje predstavljaju matricu (Slika 23.). U slučaju bijelog polja u matrici bi bio niz -1 koji bi činio matricu. Da bi se dobio izračun s desne strane potrebno je zbrajati ćelije unutar matrice. Unutar plave oznake gleda se kut koji spaja vodoravnu i okomitu crtu, a prikazuje integral određenog piksela. U slučaju plave crte kut presijecanja iznosi šest jer je ukupan zbroj ćelija vrijednost šest.



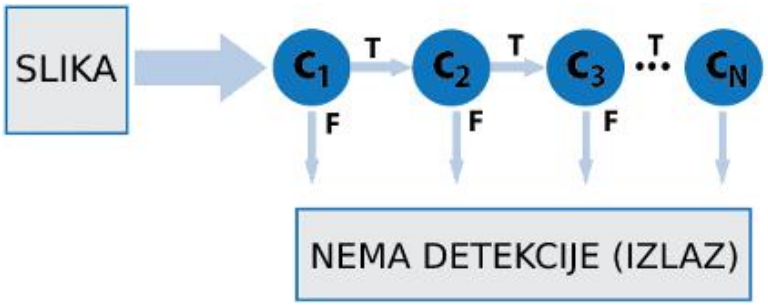
Slika 24. Prikaz integrala piksela unutar matrice

Primjerom (Slika 23.) može se izdvojiti bilo koji dio izračunatog integrala piksela. Za primjer je uzet donji dio matrice (Slika 24.) koji prikazuje četiri pravokutnika. Da bi dobili vrijednost D koriste se haar značajke za računanje 4 pravokutnika. U nastavku je prikazana formula za računanje vrijednosti.

$$D = A+(A+B+C+D) - (A+C+A+B) = 4 + (4+6+6+9) - (4+6+4+6) = 9 \quad (1)$$

Uz pomoć Adaboosta koji je algoritam za strojno učenje izabiru se najbolje haar značajke od ukupnih 160 tisuća unutar 24x24 pikesela. Da bi značajka bila relevantna izračun mora biti veći od 0, što u ovom slučaju jest (1). Da bi se ubrzala detekcija lica koriste

se kaskade (engl. cascading). U svaku kaskadu pošalje se točno određeni broj relevantnih značajki.



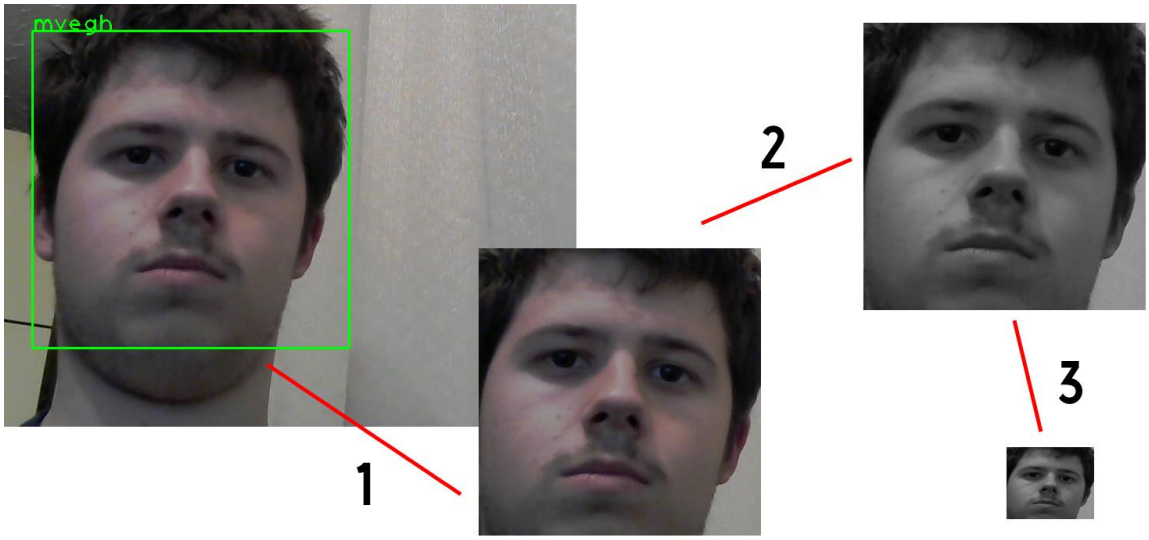
Slika 25. Prikaz kaskada

Na primjer ako je relevantnih značajki ukupno 100 u svaku kaskadu (Slika 25.) označenu s $C_{1\text{-}n}$ pošalje se 10 značajki. U slučaju da već na prvoj kaskadi nije detektirano lice, ostale se kaskade preskaču i možemo zaključiti da lice nije detektirano. Ukoliko je na prvoj kaskadi lice detektirano provjerava se druga kaskada te treća i tako sve do kraja. Ako se uspije doći do kraja putem kaskada, znači da je lice detektirano. Ako prvih nekoliko kaskada uspije detektirati lice, a sljedeća kaskada ne uspije, proces tu završava i lice nije detektirano.

9.2. Proces obrade slike

Proces obrade slike započinje nakon što je detektirano lice uz pomoć haar kaskada. Svaka slika sastoji se od matrice koja ima m stupaca i n redaka. Svaki element unutar matrice prikazuje jedan piksel na slici. Svaki piksel prikazuje određenu svjetlinu polja između 0 i 255. Svjetlina polja 0 prikazuje crnu boje, dok svjetlina polja 255 prikazuje bijelu boju. U slučaju da je slika u boji crvena, zelena, plava (engl. *red, green, blue (RGB)*) tada svaka slika nema svjetlinu piksela već se sastoji od sve tri boje. Ako se želi prikazati crna boja sve tri boje moraju sadržavati vrijednost (0, 0 ,0), a ako se želi prikazati bijela boja sve tri boje moraju sadržavati vrijednost (255, 255, 255).

Ukupan broj piksela na slici mjeri se množenjem stupca m i redaka n (mxn). Na primjer ako se slika sastoji od 100 stupaca i 100 redaka, tada slika sveukupno sadrži 10 tisuća piksela. Veću razlučivost ima ona slika koja sadrži više piksela, ali isto takvoj slici treba i više vremena za obradu. Za potrebe ovog rada koristit će se slike veličine 112 stupaca i 92 retka (112x92). Svaka slika koja pristigne na obradu koristit će 8 bitni zapis koji dopušta 256 nijansi crne boje. Svaku sliku koja pristigne na obradu potrebno je pretvoriti iz crveno, zeleno, plavo (engl. *red, green, blue (RGB)*) u tonove sive boje.



Slika 26. Proces obrade slike

Za pretvorbu crveno, zeleno, plavo (engl. *red, green, blue (RGB)*) u tonove sive boje koristi se sljedeća formula[23].

$$S = 0.299*C + 0.587*Z + 0.114*P = 0.299*255 + 0.587*255 + 0.114*0 \approx 226 \quad (2)$$

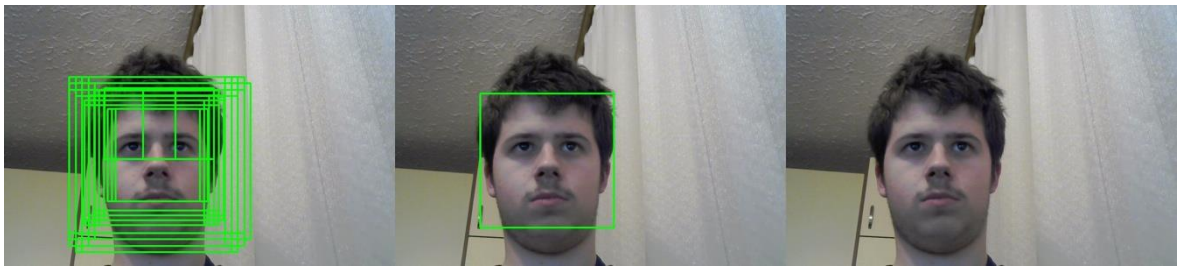
Za primjer je uzet piksel žute boje (255,255,0). Pomoću formule (2) pretvorena boja u tonovima sive boje iznos 225.93 što je približno 226 količina svjetlosti unutar piksela.

```
import numpy as np
import cv2

cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
image = cv2.imread('mario.jpg')
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

faces = cascade.detectMultiScale(gray, 1.3, 5)

for (x,y,w,h) in faces:
    cv2.rectangle(image,(x,y),(x+w,y+h),(0,255,0),2)
    crop_image = gray[y:y + h, x:x + w]
    resize_image = cv2.resize(crop_image, (112, 92))

cv2.imshow('image',resize_image)
cv2.waitKey(0)
```

Slika 27. Prikaz dobivanja slike nakon obrade

Da bi se uspješno napravio proces obrade slike (Slika 26.) potrebno je učitati haar kaskadu[Pogreška! Knjižna oznaka nije definirana.] (Slika 27.) u koju se umetne slika. Prilikom umetanja slike, ona se pretvori u tonove sive boje kako bi bilo lakše raditi. Metoda *.detectMultiScale()* prima sliku s tonovima sive boje te ostala 2 parametra. Parametar koji je označen s *1.3* zove se faktor smanjivanja i on u ovom slučaju služi da se fotografija koja je u tonovima sive boje smanji za 30%. Prema istraživanjima smanjenje slike za 30%

najoptimalnije je za detekciju lica. U slučaju da se postavi parametar *1.05* s njim bi se slika smanjila za 5% i algoritmu bi trebalo duže vremena pronaći haar značajke, ali bi se svakako povećala točnost lica jer bi slika po veličini bila bliže originalu. Parametar označen s 5 (Slika 28.) govori koliko minimalno susjednih detekcija treba biti da bi detekcija bila uspješna. Prema istraživanjima najoptimalnije je staviti parametar između 3 i 6. U slučaju da se stavi parametar manji od 3 može se dobiti preklapanje detekcije lica jer ne spaja susjede u jednu detekciju. Ako se stavi parametar veći od 6 moguće je da lice neće biti detektirano jer nema dovoljan broj susjeda.

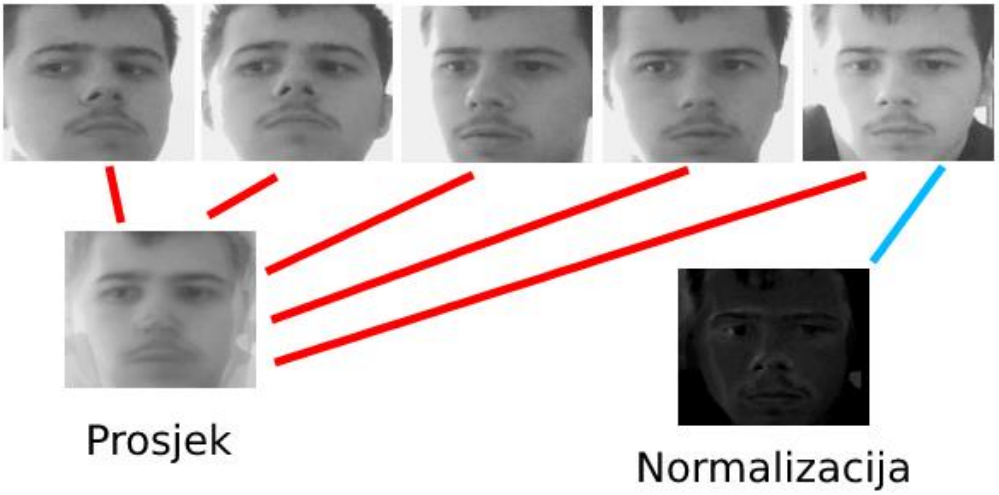


Slika 28. Prikaz minimalan broj susjeda

Unutar varijable *faces* nalaze se sva detektirana lica s njihovim kordinatnim osima (x,y), te visinom i širinom slike. Uz pomoć metode *.rectangle()* iscrtava se zeleni pravokutnik nad licem sa širinom rubova 2 piksela. Unutar varijable *crop_images* slika sa sivim tonovima izrezuje se unutar zelenog okvira te se postavlja na zadanu širinu i visinu koje su u ovom slučaju (112x92). Na kraju se uz pomoć metode *imshow()* slika ispisuje korisniku na ekran(Slika 27.).

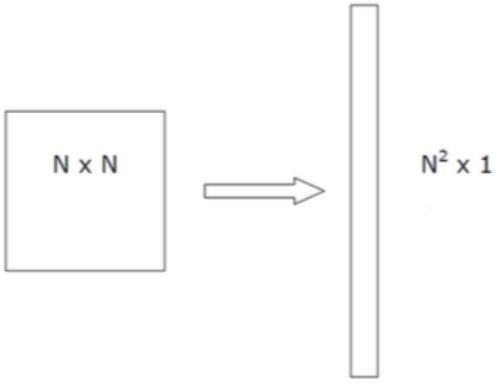
9.3. Proces prepoznavanja lica

Za proces prepoznavanja lica poželjno je imati što više slika određene osobe. Sve slike moraju biti istih dimenzija što znači imati isti broj stupaca m i redaka n (mxn) te da je glava osobe na slici centrirana. Zbog jednostavnosti analiziranja matrica, sve slike trebaju biti u tonovima sive boje.



Slika 29. Prikaz baze slika, njegovog prosjeka, te normalizacije

Za početak možemo zamisliti da imamo dva skupa podataka. Jedan skup podataka može se zvati “mario“, a drugi skup može se zvati “pero“. Unutar jednog skupa podataka nalazi se mnogo slika (Slika 29.). Za primjer je uzeto 100 slika istih dimenzija, da bi se olakšalo razumijevanje prepoznavanja zamislimo da je svaka slika (50x50) (nxn), što znači da se jedna takva slika sastoji od 2500 piksela odnosno dimenzija. Svaku sliku[24] unutar skupa podataka potrebno je staviti u jednodimenzionalni vektor jer prepoznavanje lica može samo raditi s jednodimenzionalnim vektorom(Slika 30).



Slika 30. Prikaz jednodimenzionalnog vektora

Svaku sliku sada možemo zamisliti kao jedan štapić gdje su spojene kolone i redovi u isti niz. Niz takvih štapića čine vektore lica u prostoru (engl. *face vector space*).

```
import numpy as np
n_x_n = np.array([[1, 2], [3, 4]])
n_square = n_x_n.flatten()

print(n_square) #([1, 2, 3, 4])
```

Kôd 11. Pretvaranje slike u jednodimenzionalni vektor

Sljedeći korak je izračunavanje prosjeka nad skupom slika (Slika 29.). Uz pomoć prosjeka znamo sve zajedničke stavke jednog skupa podataka.

```
import numpy as np
images_from_mario_dataset = [n_square]
average = np.mean(images_from_mario_dataset, axis = 0
```

Kôd 12. Dobivanje srednje vrijednosti unutar skupa

Da bi dobili unikatne značajke vezane za svako lice unutar skupa potrebno je nad njim napraviti normalizaciju. Normalizacija je proces oduzimanja od trenutne slike, prosjek svih slika u skupu.

```
import numpy as np
images_from_mario_dataset = [n_square]
average = np.mean(images_from_mario_dataset, axis = 0)
for image in images_from_mario_datase
      normalization = image - average
```

Kôd 13. Normalizacija svake slike unutar skupa

Kako bi mogli izračunati eigenvektore potrebno je izračunati kovarijancu matrice.

```
import numpy as np
covariance = np.dot(normalization, normalization.T)
```

Kôd 14. Kovarijanca matrice

Za potrebe računanja kovarijance moramo imati zasebne značajke za svako lice vektora (engl. *face vector*). Unutar (Kôd 14.) varijabla normalization je zapravo (N^2 x M). Oznaka M označava broj vektora lica (štapića) u prostoru (engl. *face vector space*).

$$C = normalization*normalization^T = (N^2 \times M) * (M \times N^2) = N^2 \times N^2 = 2500 \times 2500 = 6250000 \quad (3)$$

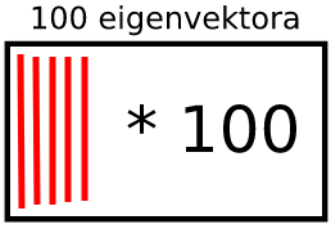
Broj 2500 (3) dobije se kada se pomnoži stvarna veličina slike (50x50) koja prikazuje broj dimenzija slike odnosno piksele.

Slika 31. Prikaz kovarijance matrice u visokoj dimenziji

Svaki eigenvektor unutar eigenvektora (Slika 31.) ima jednu dimenziju. Za dobivanje eigenfaces moramo se spustiti u nižu dimenziju radi lakoće računanja. Izračun se može odraditi i u višoj dimenziji, ali za taj poduhvat računalu treba mnogo vremena te se zato prebacuje u nižu dimenziju. Eigenfaces mora biti manji ili jednak broj slika u skupu podataka K <= M

Za prebacivanje u nižu dimenziju koristi se kovarijance matrice s formulom (4) koja će smanjiti dimenziju.

$$C = \text{normalization}^T * \text{normalization} = (M \times N^2) * (N^2 \times M) = M^2 \times M^2 = 100 \times 100 = 10000 \quad (4)$$

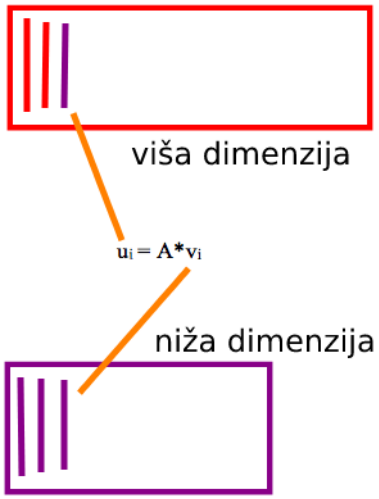


Slika 32. Prikaz kovarijance matrice u nižoj dimenziji

U nižoj dimenziji (Slika 32.) smanjeno je za 624000 eigenvektora, čime se skratilo vrijeme računanja. Za prebacivanje iz niže dimenzije u višu dimenziju (Slika 33.), svaki eigenvektor iz niže dimenzije mora biti množen s vrijednosti normalizacije (N^2 x M).

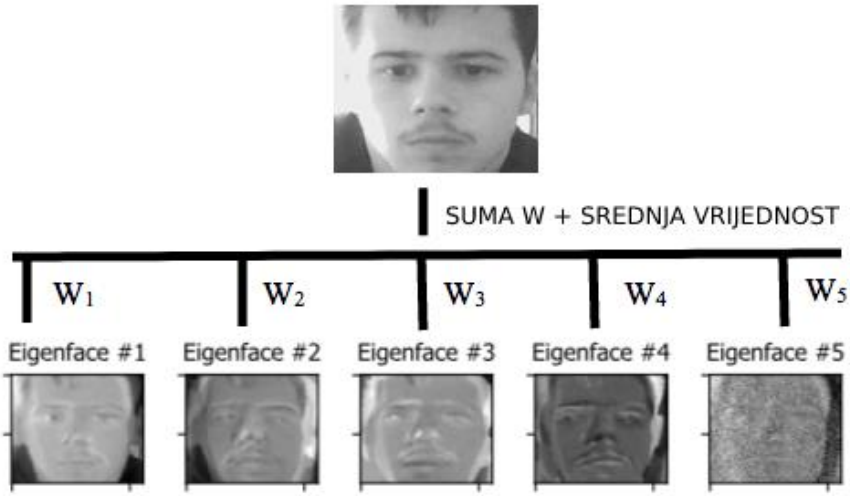
$$u_i = A * v_i \quad (5)$$

Unutar formule (5) u_i označava višu dimenziju, A označava vrijednost normalizacije (N^2 x M) te v_i svaki vektor unutar niže dimenzije.



Slika 33. Prikaz prebacivanja eigenvektora iz niže dimenzije u višu dimenziju

Prebačen vektor iz niže dimenzije (Slika 33.) u višu dimenziju prikazuje određeno lice (eigenface) unutar skupa podataka određene osobe.



Slika 34. Prikaz eigenface u skupu podataka

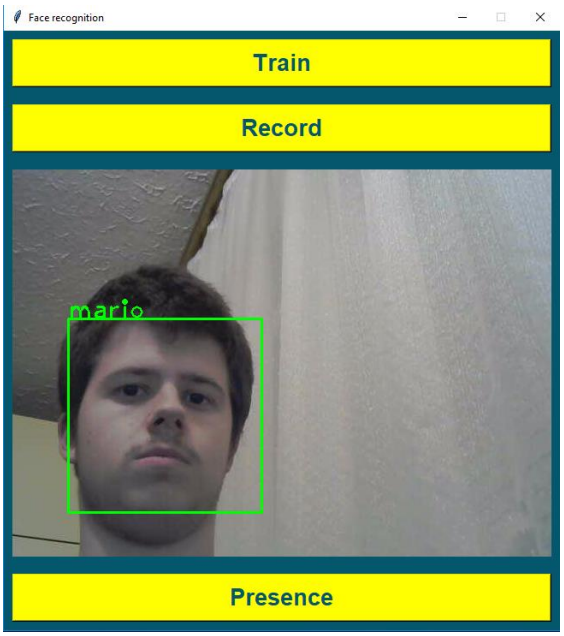
Svaki eigenface (Slika 34.) govori koliki postotak težine vektora (W_n) doprinosi stvaranju glavnog lica. Zbrojem svih težina vektora (W_n) + srednja vrijednost dobije se glavna slika. Srednja vrijednost dodaje se jer smo na početku prije svih računanja maknuli srednju vrijednost te je sada opet dodajemo. Srednja vrijednost je bitna jer sadrži značajke svih lica.

9.4. Proces prepoznavanja nepoznatog lica

Nepoznato lice koje ulazi u sustav (n x n) treba pretvoriti u vektor lica (engl. *face vector*). (n^2 x 1) (Kôd 11.). Da bi izdvojili značajke samo tog lica potrebno ga je normalizirati. Iz cijelog skupa podataka za određenu osobu potrebno je izračunati srednju vrijednost svih slika kako bi dobili iste značajke za cijeli skup (Kôd 12.). Normalizacija je proces oduzimanja od jedne slike, srednju vrijednost skupa svih slika određene osobe (Kôd 13.). Nepoznato lice treba prikazati kao kombinaciju eigenface unutar prosječnog skupa podataka koji nam služi za prepoznavanje te se izračuna težina vektora za svaki eigenface. Za nepoznato lice potrebno je izračunati udaljenost između težinskog vektora i svih vektora unutar jednog skupa podataka za određenu osobu, na primjer “mario“ (Slika 35.). Ako je udaljenost manja od zadanog praga, tada je osoba prepoznata. Ukoliko je udaljenost veća od zadanog praga, osoba biva nepoznata. U primjeru (Slika 35.) može se vidjeti da je prag (engl. *threshold*) za poznato lice 5000.

```
for i in range(len(self.project_image_list)):
    result = np.linalg.norm(input_image - self.project_image_list[i])
    if(result < min_result):
        min_result = result
        person_target = self.person_name[i]
if(min_result > 5000):
    person_target = "Unknown"
```

Slika 36. Prepoznavanje nepoznatog lica



Slika 35. Prikaz desktop aplikacije za prepoznavanje lica

10. Implementacija

Unutar ovog poglavlja prikazivat će se ključni dijelovi kako bi aplikacija radila uspješno.

10.1. Proces prikupljanja slika za treniranje modela

```
def read_all_images(self, path):
    images = []
    images_person = []
    flatten_matrix_list = []
    for root, folders_in_root, files in os.walk(path):
        for folder in folders_in_root:
            root_with_folder = os.path.join(root, folder)
            matrix_for_each_person = []
            for file in os.listdir(root_with_folder):
                #for mac
                if file != ".DS_Store":
                    try:
                        im = Image.open(os.path.join(root_with_folder, file))
                        images.append(np.asarray(im, dtype=np.uint8))
                    except IOError:
                        print("IOError", IOError.args[0], IOError.args[1])
                    except:
                        print("Error", sys.exc_info()[0])
                        raise
                    matrix_for_each_person.append(np.asarray(im, dtype=np.uint8))
            flatten_matrix = np.array([image.flatten() for image in matrix_for_each_person], dtype='f')
            flatten_matrix_list.append(flatten_matrix)
            images_person.append(folder)

    return images, images_person, flatten_matrix_list
```

Slika 37. Proces prikupljanja slika

Proces prikupljanja slika (Slika 37.) započinje tako što se u metodu - pročitaj sve slike (engl. *read_all_images*) pošalje naziv glavne mape koja sadrži ostale mape s imenima osoba. Unutar glavne mape prođe se po svim unutarnjim mapama te se spaja naziv glavne mape s mapom trenutne osobe. Unutar tako spojene mape prođe se kroz sve datoteke (slike) te se svaka od njih prvo mora pretvoriti u oblik matrice pomoću metode otvori (engl. *open*). Takva matrica sprema se u polje s 8 bitnim zapisom što označava 256 nijansi kod slike s tonovima sive boje. Unutar polja slike (engl. *images*) nalaze se slike svih osoba, nevažno u kojim su mapama. Sve slike za jednu osobu također se spremaju u varijablu matrice za svaku osobu (engl. *matrix for each person*) te nam ta varijabla služi kako bi se iz nje moglo stvoriti jednodimenzionalno polje (N^2 x 1) za sve slike određene osobe. Također, varijabla matrice za jednu osobu dodaje se u listu koja će imati toliko podataka koliko osoba ima u sustavu. Na kraju je potrebno imati još jedno polje u koje će se spremati imena za svaku osobu. Sam proces završava tako što metoda vrati sve slike (n x n), nazive svih osoba te listu slika za svaku osobu u obliku jednodimenzionalnog polja (N^2 x 1).

10.2. Prikupljanje podataka o mobilnom uređaju i slanje na web servis

1.)

```
@Override
protected Void doInBackground(String... strings) {
    OkHttpClient client = new OkHttpClient();

    RequestBody body = new FormBody.Builder()
            .add( name: "token", strings[0])
            .add( name: "user", strings[1])
            .add( name: "group_name", strings[2])
            .build();

    Request request = new Request.Builder()
            .url("RegisterDevice.php")
            .post(body)
            .build();

    try {
        client.newCall(request).execute();
    } catch (IOException e) {
        e.printStackTrace();
    }
    return null;
}
```

2.)

```
if($_SERVER['REQUEST_METHOD']=='POST'){

    $token = $_POST['token'];
    $user = $_POST['user'];
    $groupid = $_POST['group_name'];

    $db = new DbOperation();

    $result = $db->registerDevice($user, $token, $groupid);
}
```

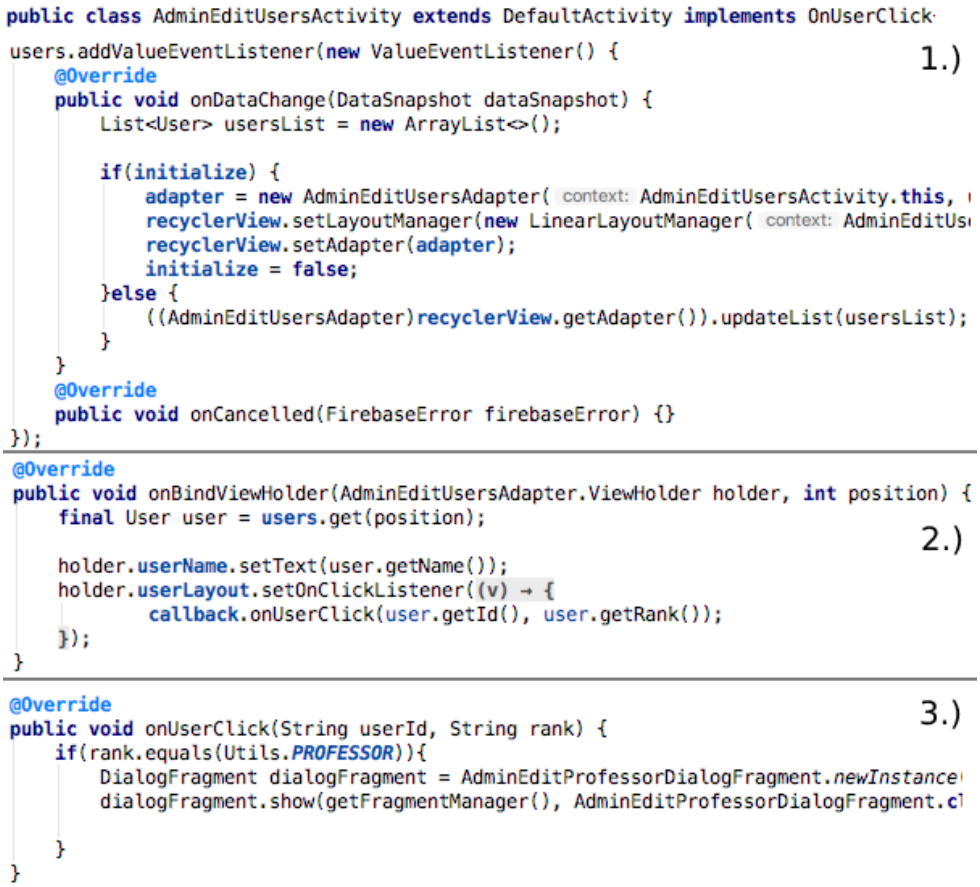
3.)

```
public function registerDevice($user, $token, $group){
    if(!$this->isUserExist($user)){
        $stmt = $this->con
        ->prepare("INSERT INTO users (user, token, group) VALUES (?,?,?) ");
        $stmt->bind_param("sss",$user,$token, $group);

        if($stmt->execute()){
            return 0;
        }
        return 1;
    ...
```

Slika 38. Prikupljanje podata o mobilnom uređaju

Proces prikupljanja podataka (Slika 38.) započinje kada se student prijavi unutar Android aplikacije. Android aplikacija asinkrono šalje trenutni token od Firebase i grupu iz koje student dolazi na web servis napisan u PHP programskom jeziku. Unutar PHP web servisa prikupljaju se podaci pristigli s Androida te se šalju u metodu registriraj uređaj. U slučaju da korisnik ne postoji u MySQL bazi podataka unutar nje se dodaje korisnik s podacima token i grupa. Ako već slučajno student postoji u bazi, a promjeni mu se token ili grupa, unutar web servisa će se samo promijeniti određeni parametar. Registracija studenta je nužna kako bi profesor mogao slati obavijesti studentima. Također kod prepoznavanja lica da bi svaki student prilikom ulaska u učionicu mogao dobiti obavijest na svoj mobilni uređaj o prisustvu, potrebna je registracija uređaja. Registracija uređaja obavlja se u pozadini tako da to student neće ni primijetiti.

10.3. Prikaz podataka na Android aplikaciji

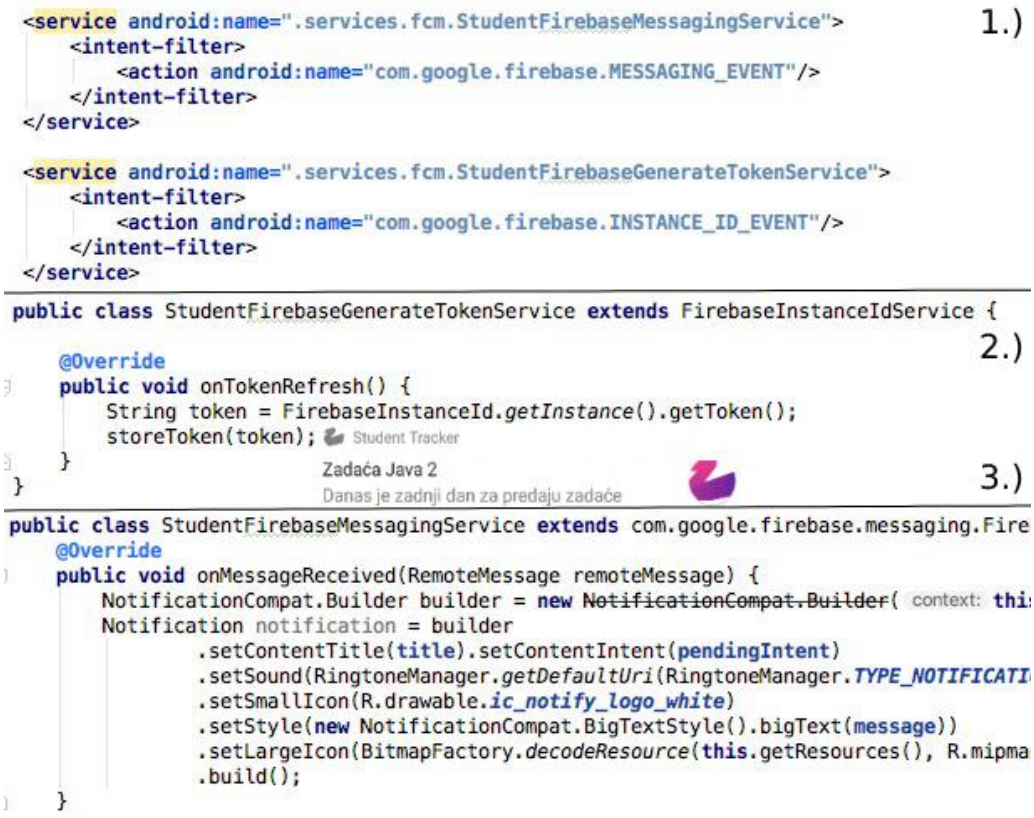


Slika 39. Prikaz podataka na Android aplikaciji

Za prikaz podataka (Slika 39.) unutar aplikacije koristi se Firebaseova metoda *.addValueEventListener()*. Uz pomoć ove metode aplikacija sluša sve događaje koji se događaju na bazi bilo da je unesen, izbrisan ili uređen element. Pri prvom pozivu unutar metode treba se istancirati adapter koji prikazuje podatke. Kako bi korisnik uspješno vidio podatke adapter mora biti dodan u reciklažni pogled (engl. *recycler view*). Pri istanciranju adaptera može se poslati i slušatelj (engl. *listener*) koji osluškuje događaj kada korisnik klikne. Pri samom vrhu (Slika 39.) može se vidjeti događaj "na korisnikov klik" (engl. *on user click*) gdje uz pomoć implementacije sučelja možemo osluškivati događaj koji je prikazan na dnu (Slika 39.). Unutar adaptera nalazi se metoda *onBindView()* koja služi za spajanje XML dijela s programskim dijelom u Javi. Klikom na određenog korisnika aplikacija istog trena šalje podatke iz povratnog poziva (engl. *callback*) natrag korisniku u metodu *onUserClick()* koja je prikazana na dnu (Slika 39.). Kada se promjene podatci, doda novi podatak ili pak izbriše podatak *.addValueEventListener()* će se ponovo pozvati.

Pri novom pozivu adapter se više neće instancirati već će se prosljediti nova lista s napunjenim podacima. Novi podaci će istog trena u metodi *.updateList()* zamijeniti stare podatke te će se korisniku na ekran prikazati novi osvježeni podaci.

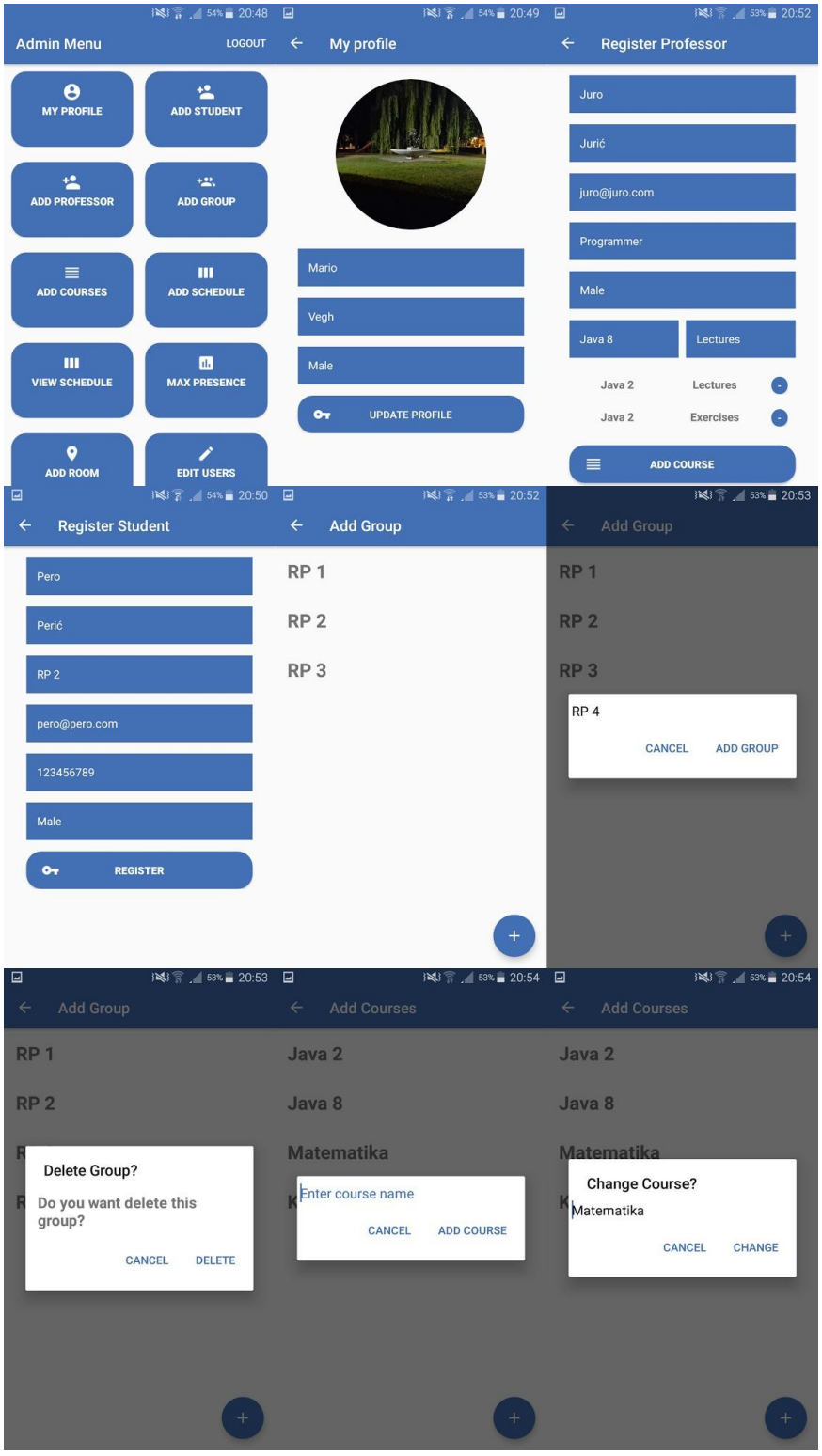
10.4. Prikaz obavijesti na mobilnom uređaju



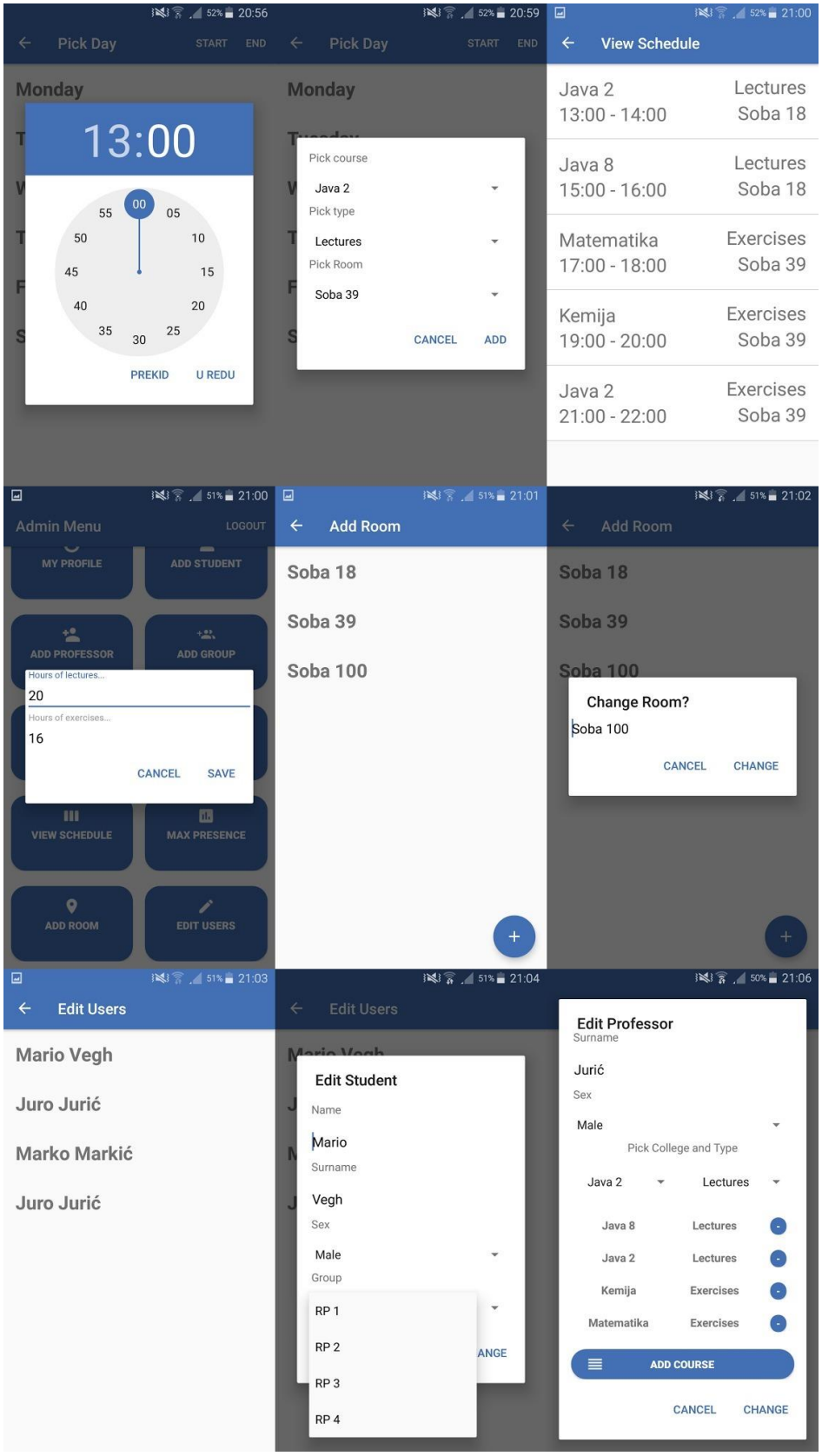
Slika 40. Prikaz obavijesti na mobilnom uređaju

Obavijesti na mobilni uređaj šalju se putem PHP web servisa. Unutar mobilne aplikacije postoje dva servisa (Slika 40.). Prvi servis služi za primanje poruka, dok drugi služi za generiranje tokena putem Firebase. Unutar prvog servisa kada pristigne poruka poziva se metoda *onMessageReceived()*. Unutar metode *onMessageReceived()* kreira se izgled obavijesti kako će izgledati na mobilnom uređaju. Također na slici (Slika 40.) je vidljivo kako obavijest izgleda na mobilnom uređaju. Prilikom pristizanja obavijesti na mobilnom uređaju pristiže i zvukovni znak koji obavještava primatelja o primitku poruke. Unutar drugog servisa (Slika 40.) u metodi *onTokenRefresh()* postavlja se token za svakog studenta koji pristupi aplikaciji. Token koji je kreirao Firebase proslijeđuje se dalje na web servis za registraciju uređaja. Novi token generira se samo u slučaju ako se student ponovo prijavi u aplikaciju, tada se ponovo radi poziv prema web servisu i stari token se zamjenjuje novim.

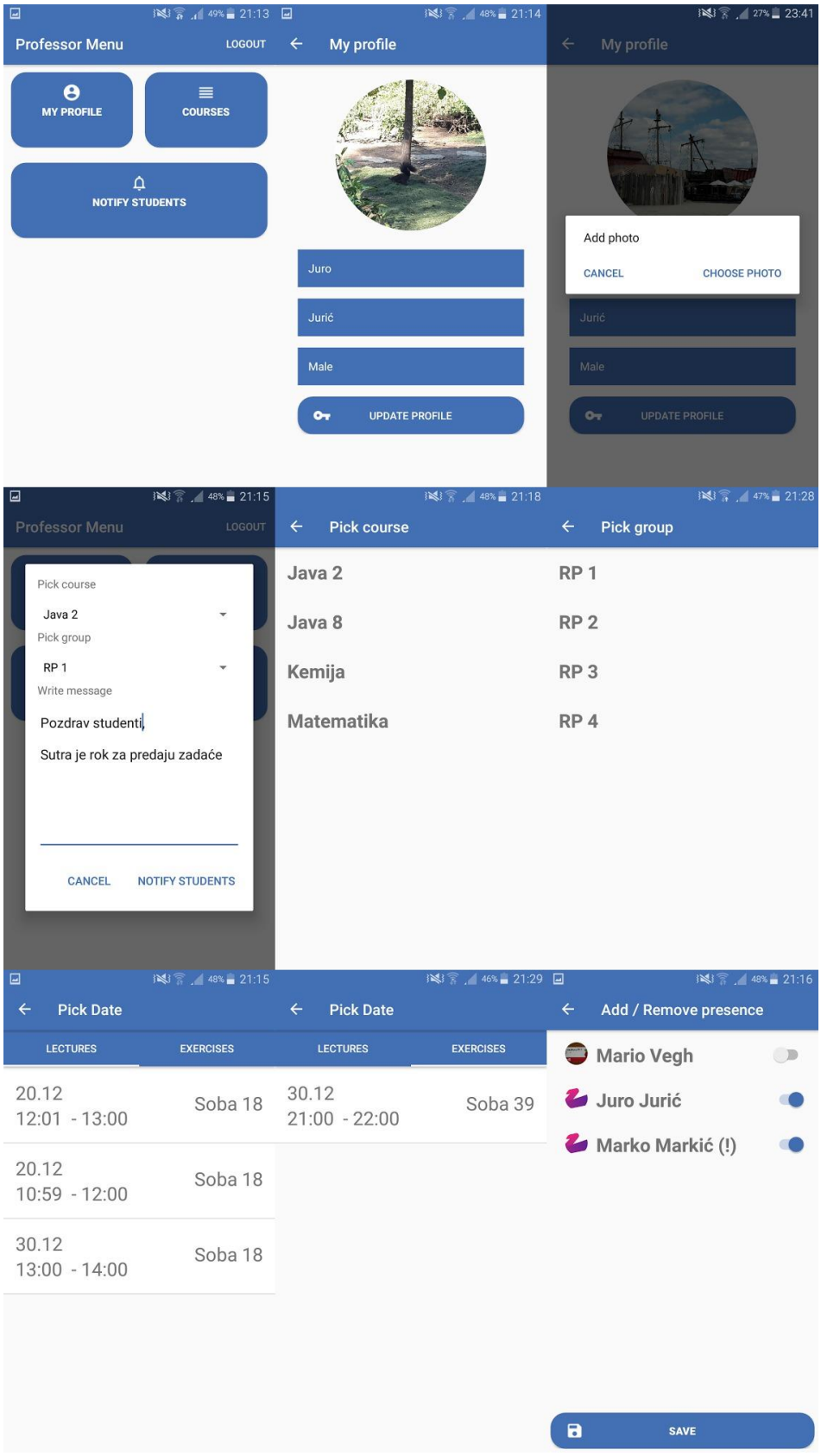
10.5. Korisničko sučelje Android aplikacije



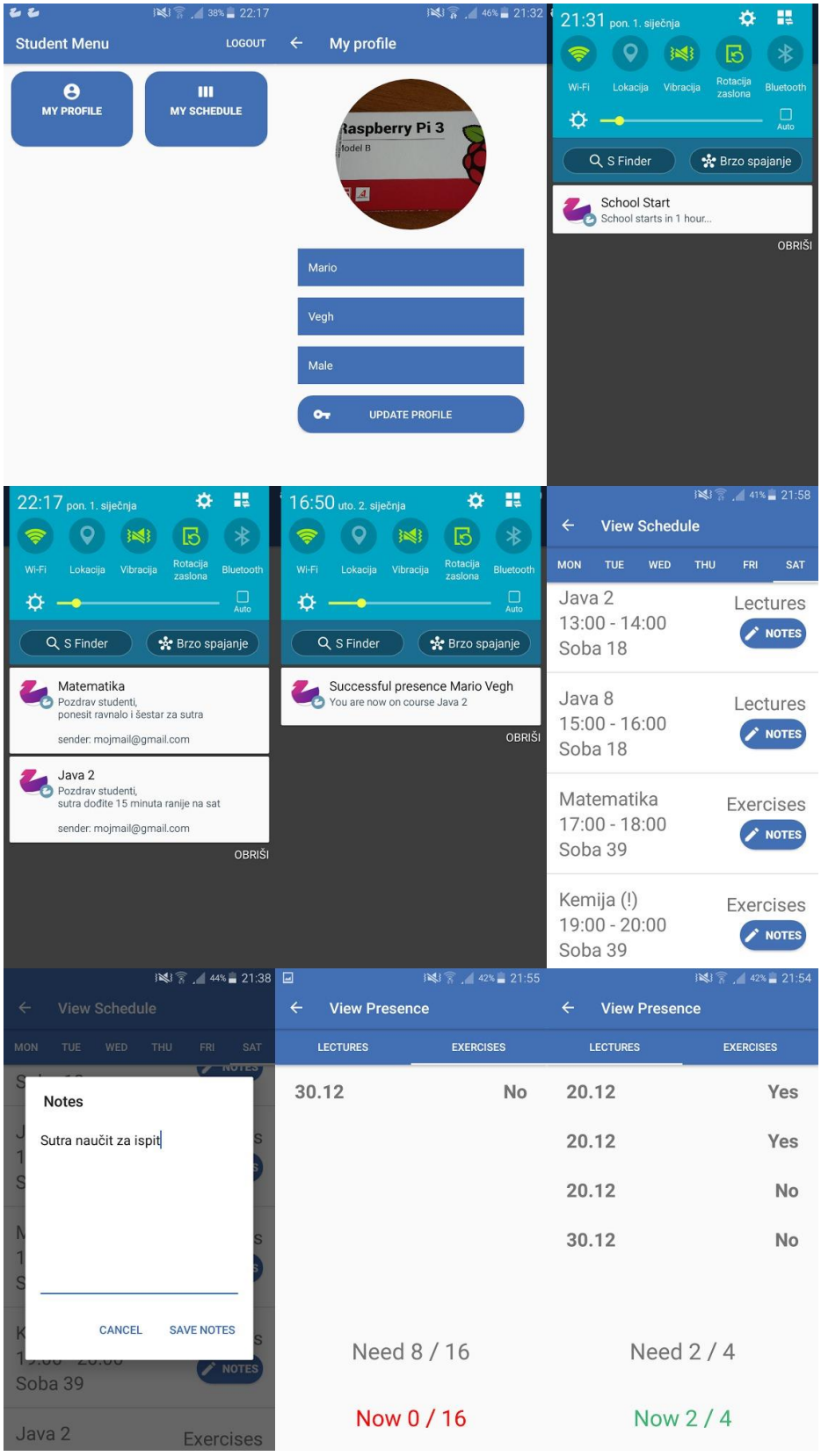
Slika 41. Sučelje role administrator (1)



Slika 42. Sučelje role administrator (2)



Slika 43. Sučelje role profesor



Slika 44. Sučelje role student

Zaključak

Aplikacija za praćenje prisutnosti studenata na nastavi uspješno je izrađena i spremna je za probno korištenje unutar obrazovne ustanove. Trenutno aplikacija još nije testirana u svom pravom okruženju te se ne može procijeniti koliko je uspješna u praksi. Testiranje se vršilo na četiri studenta i tada je uspješno radio segment Android aplikacije i komunikacija putem obavijesti. Segment detekcije i prepoznavanja lica pokazao se manje uspješnim u različitim okolinama.

Nedostatak trenutnog rješenja unutar Android aplikacije su semestri, praćenje zauzetosti profesora i učionice pri kreiranju novog rasporeda. Android aplikacija prati studente za jedan semestar. U slučaju da se želi nastaviti pratiti studenta i za sljedeći semestar aplikaciju je potrebno nadograditi. Pri kreiranju rasporeda može se desiti preklapanje dotičnog profesora ili učionice za određeni termin. Sustav ne dozvoljava vremenska preklapanja za svaku grupu.

Segment detekcije i prepoznavanja lica pokazao se kao najslabija točka ovog sustava. Najtočniji rezultati se dobivaju ako student nepokrivenim licem gleda u kameru te ako je osvjetljenje učionice istog intenziteta u toku dana ili noći. Na mjestu gdje se radilo treniranje lica, bit će najveća točnost prepoznavanja lica, zbog kuta padanja svjetlosti. Pri treniranju lica potrebno je imati što više slika određene osobe. Za postizanje najboljih rezultata za jednu osobu najbolje je imati između 400 i 600 slika.

Vremensko trajanje od upisa podatka u bazu podataka pa do ispisa korisniku na ekran je trenutno što daje korisniku privid kreiranja podatka u stvarnom vremenu. U slučaju da korisnik ostane bez mreže podaci se spremaju lokalno. Prilikom pristupanja mreži aplikacija šalje podatke prema poslužitelju. Plan za sljedeću fazu rada je kreiranje više semestara, rješenje zauzetosti profesora i učionice, praćenje dolazaka studenata na nastavu putem otisaka prstiju te kategoriziranje kolegija prema godini.

Popis kratica

CRUD	Create, Retrive, Update, Delete	Kreiranje, dohvaćanje, ažuriranje, brisanje
IP	Internet protocol	Internet protokol
API	Application programming interface	Aplikacijsko programsko sučelje
JSON	Javascript object notation	Javascript notacija objekata
SDK	Software development kit	Razvojni softverski paket
LAMP	Linux, Apache, MySQL, PHP	Linux, Apache, MySQL, PHP
PHP	PHP Hypertext Preprocessor	PHP hipertekst pred-procesor
SQL	Stuctured Query Language	Strukturirano upitni jezik
CSV	Comma-separated values	Vrijednosti odvojene zarezom
PDF	Portable Document Format	Prijenosno formatirani dokument
XML	EXtensible Markup Language	Jezik za označavanje podataka
FCM	Firebase Cloud Messaging	Firebase poruke u oblaku
HTTP	Hypertext transfer protocol	Protokol prijenosa informacija
cURL	command-line Uniform Resource Locator	Naredbeno linijski usklađeni lokator sadržaja
URL	Uniform Resource Locator	Usklađeni lokator sadržaja
GUI	Graphical user interface	Grafičko korisničko sučelje
RGB	Red, Green, Blue	Crveno, zeleno, plavo

Popis slika

Popis tablica

Popis kôdova

Literatura

[1] Grupa autora, Algebra, Izrada aplikacija za mobilne uređaje, Zagreb 2013.
[2] https://firebase.google.com/products, prosinac 2017.
[3] https://firebase.google.com, prosinac 2017.
[4] Isuru Madusanka, Rukman Bernard Busy programmers guid to Firebase with Android, rujan 2017.
[5] https://firebase.google.com/docs/database/android/structure-data, rujan 2017.
[6] https://firebase.google.com/docs/storage/android/handle-errors, rujan 2017.
[7] https://firebase.google.com/docs/auth/android/firebaseui, rujan 2017.
[8] Antiono Lopez, Learning PHP 7, ožujak 2016.
[9] https://code.tutsplus.com/tutorials/relational-databases-for-dummies--net-30244, prosinac 2017.
[10] https://bitnami.com/stack/lamp, prosinac 2017.
[11] https://docs.phpmyadmin.net/en/latest/intro.html, prosinac 2017.
[12] Hasin Hyder, Object-Oriented Programming with PHP5, prosinac 2007.
[13] https://medium.com/@lazlojuly/what-is-a-restful-api-fabb8dc2afeb, prosinac 2017.
[14] https://www.json.org, prosinac 2017.
[15] https://www.getpostman.com, prosinac 2017.
[16] https://firebase.google.com/docs/cloud-messaging/send-message, prosinac 2017.
[17] http://square.github.io/otto, prosinac 2017.
[18] Bhaskar Chaudhary, Tkinter GUI Application Development Blueprints, studeni 2015.
[19] https://pypi.python.org/pypi/python-firebase/1.2, prosinac 2017.
[20] https://opencv.org/about.html, prosinac 2017.
[21] Preteek Joshi, Vinicius Godoy, David Millan Escriva, OpenCV by Example, siječanj 2016.
[22] https://docs.opencv.org/3.3.0/d7/d8b/tutorial_py_face_detection.html, prosinac 2017.
[23] https://docs.opencv.org/3.1.0/de/d25/imgproc_color_conversions.html, prosinac 2017.
[24] https://stephenpaulraj.com/face-detectionrecognition-eigenfaces, prosinac 2017.
[25] https://mmikowski.github.io/the_lie, prosinac 2017.
[26] https://www.rosehosting.com/blog/what-is-curl, siječanj 2018.
[27] https://pypi.python.org/pypi/opencv-python, prosinac 2017.
[28] Joseph Howse, OpenCV for Secret Agents, siječanj 2016.
[29] Joseph Howse, OpenCV Computer Vision with Python, travanj 2013.
[30] Jan Erik Solem, Programming Computer Vision with Python, lipanj 2012.