Detekcija i raspoznavanje lica unutar iOS aplikacije

Barbarić, Nikola

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

DETEKCIJA I RASPOZNAVANJE LICA UNUTAR iOS APLIKACIJE

Diplomski rad

Nikola Barbarić

Osijek, 2021.

SADRŽAJ

1. UVOD

Tema ovog diplomskog rada je izrada mobilne aplikacije za detekciju i prepoznavanje lica, te vršenje evidencije na osnovu prepoznatih lica. Na osnovu prepoznatih lica vršit će se evidencija dolaznosti studenata na nastavu, te bi diplomski rad imao primjene i u stvarnom svijetu. Postoje neka slična rješenja i ona od njih koja je vrijedno spomenuti su prikazana u drugom poglavlju. Budući da je iOS platforma dosta zatvorenijeg tipa za izradu se koristi službeno programsko okruženje Xcode, te programski jezik Swift. Prilikom izrade mobilne aplikacije bit će pokriveni njezini najvažniji dijelovi i opisani u posebnim poglavljima. Dolaznost studenata na nastavu je još uvijek obvezna na većini sveučilišta kako u svijetu tako i u Hrvatskoj. Većina profesora se trudi osmisliti što efikasnije metode provjere prisutnosti koje je teško zaobići. Budući da se lice ne može krivotvoriti za razliku od potpisa i ostalih načina provjere, ova metoda ima veću jedinstvenost od potpisa. Budući da se objava na službenom App Storeu plaća, te da bi se aplikacija kao produkt mogla prodavati sveučilištima gdje bi se interno koristila, ona kao takva neće biti objavljena nigdje, ali će sav kod biti vidljiv u prilozima, dok će najvažniji dijelovi biti prikazani u posebnim poglavljima.

1.1. Zadatak diplomskog rada

Istražiti i opisati algoritme za detekciju i raspoznavanje lica. Opisati najčešće primjene tih tehnika. Ispitati izazove kod detekcije raspoznavanja lica, tehnološke, zakonske i etičke. Istražiti tehnike detekcije i raspoznavanja lica dostupne na iOS platformi. Istražiti i opisati Appleove biblioteke za strojno učenje i računalni vid. Implementirati iOS aplikaciju s primjenom detekcije i raspoznavanja lica. Testirati implementaciju na proizvoljnim testnim slučajevima. Prezentirati i objasniti dobivene rezultate i aplikaciju.

2. PREGLED POSTOJEĆIH RJEŠENJA

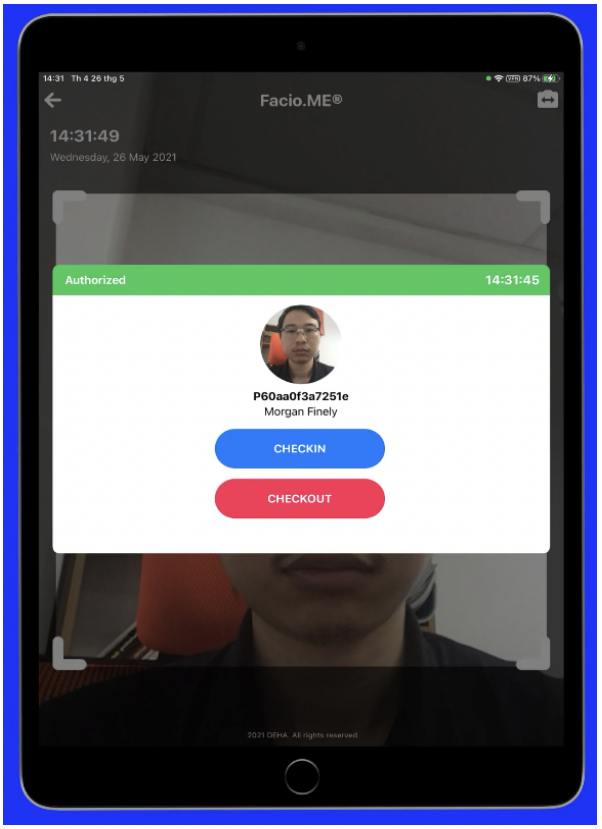
U nastavku su prikazana neka od postojećih rješenja. Postoji još rješenja, ali neka od njih nije bilo moguće testirati zbog toga što su napravljena za internu upotrebu, te imaju ograničen pristup.

2.1. Postojeće aplikacije

Izdvojene su neke od najpopularnijih aplikacija koje se mogu preuzeti u službenoj trgovini.

Facio.ME Kiosk

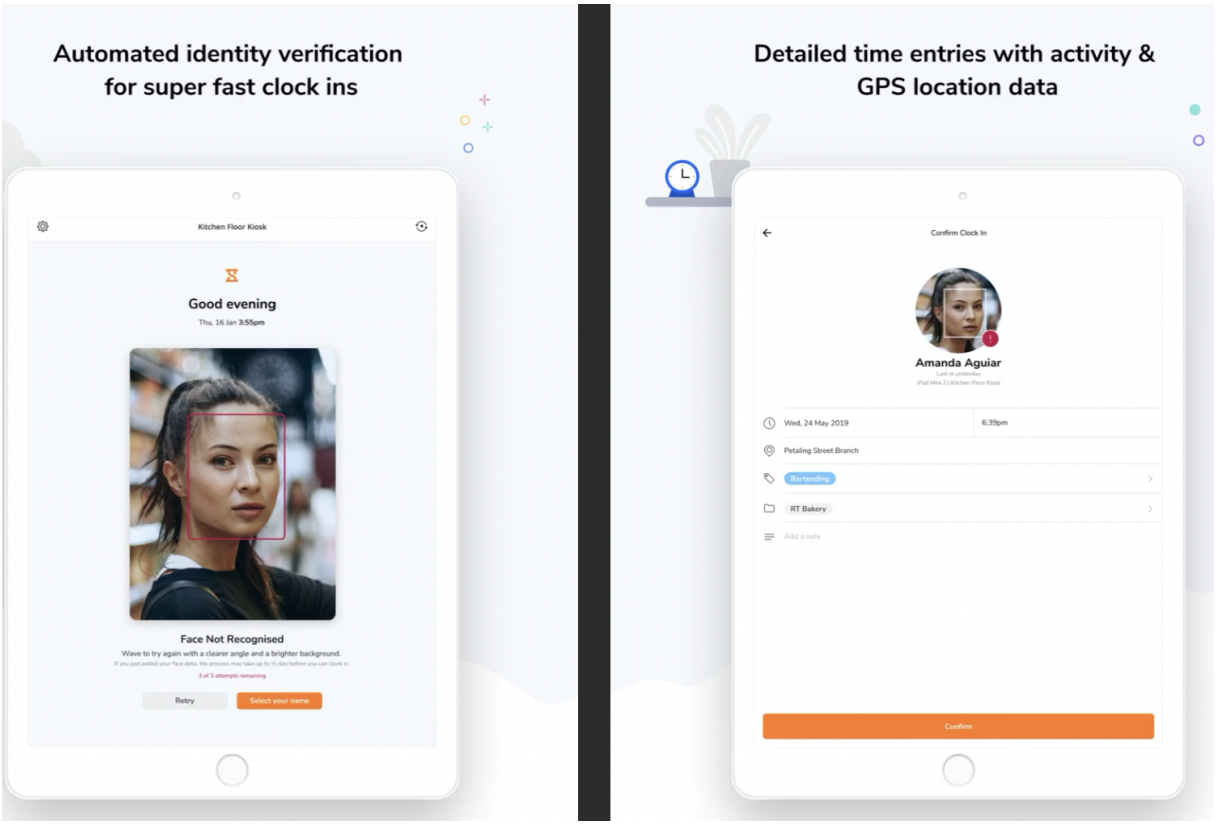
Ova aplikacija je zamišljena za vođenje evidencije nazočnosti zaposlenika. Podatci se šalju na oblak, a sprema se datum i vrijeme kada se zaposlenik prijavio, odnosno odjavio s posla. Aplikacija je jednostavna za korištenje, međutim predviđena je samo za iPad. [1]



Slika 2.1. Facio.ME Kiosk aplikacija

Jibble 2: Employee Time Clock

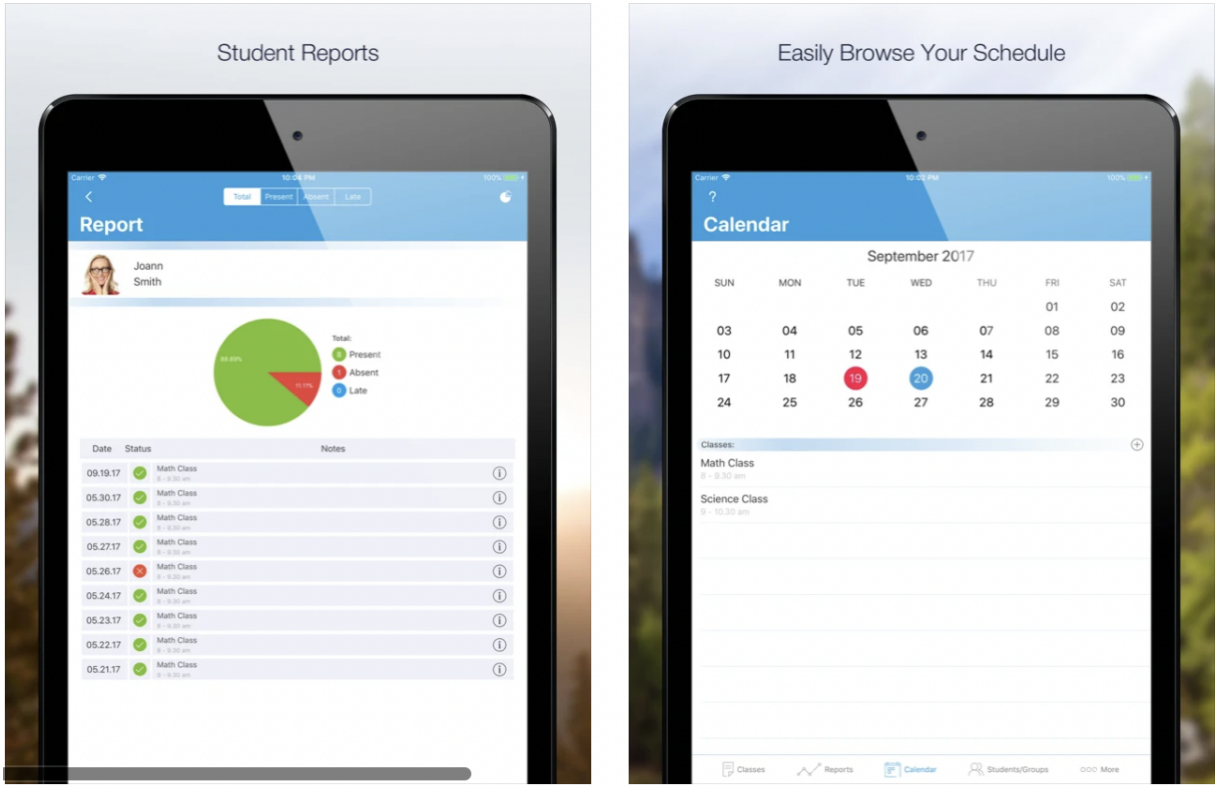
Jibble 2 je aplikacija dizajnirana za iPad. U odnosu na prošlu ima ljepši dizajn i veći izbor funkcionalnosti. Nudi spremanje vremena prijave i odjave zaposlenika, kao i zapis lokacije na kojim se prijava odnosno odjava dogodila. Osim navedenog nudi mogućnost slanja obavijesti putem aplikacije ili putem elektroničke pošte nadređenom ako se neki od zaposlenika nije prijavio odnosno odjavio u određeno vrijeme. [2]



Slika 2.2. Izgled Jibble 2 aplikacije

Alora App

Ova aplikacija postoji u besplatnoj i plaćenoj verziji, te je najsličnija aplikaciji koja je tema ovog diplomskog rada. Besplatna verzija omogućuje dodavanje neograničenog broja predavanja, kao i studenata. Omogućuje praćenje dolaznosti, dodavanja bilješki, kao i sinkronizaciju između uređaja. Plaćena verzija nudi pametne izvještaje, izvoz podataka u pdf ili csv formatu, te kolaboraciju u timu. [3]



Slika 2.3. Alora aplikacija

2.2. Postojeće metode za raspoznavanje lica

U nastavku su navedene neke od trenutno najpopularnijih metoda za raspoznavanje lica. Za neke algoritme je potrebna veća računalna moć pa se često slike šalju na oblak gdje se obrađuju. Zbog privatnosti i enkripcije to uvijek nije moguće ni poželjno. Upravo zbog toga je i Apple od verzije iOS 10 počeo koristiti duboko učenje umjesto dotadašnjeg Viola-Jones algoritma. Gruba podjela algoritama za raspoznavanje lica je na one koji koriste 2d prikaz i one koji koriste 3d prikaz.

Viola - Jones algoritam

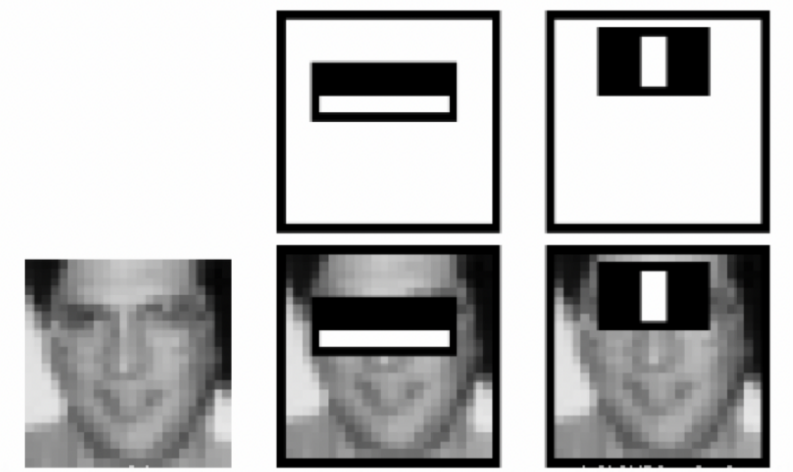
Razvili su ga Paul Viola i Michael Jones 2001. godine. Baziran je na strojnom učenju. Uz pomoć velikog broja pozitivnih i negativnih slika uvježbava se model koji se koristu na drugim slikama. Na slici 2.4. je prikazan pseudokod ovog algoritma.

Algorithm: Viola-Jones Face Detection Algorithm

```
Input: original test image
Output: image with face indicators as rectangles
for i ← 1 to num of scales in pyramid of images do
    Downsample image to create image_i
    Compute integral image, image_ii
    for j ← 1 to num of shift steps of sub-window do
        for k ← 1 to num of stages in cascade classifier do
            for l ← 1 to num of filters of stage k do
                Filter detection sub-window
                Accumulate filter outputs
            end for
            if accumulation fails per-stage threshold then
                Reject sub-window as face
                Break this k for loop
            end if
        end for
        if sub-window passed all per-stage checks then
            Accept this sub-window as a face
        end if
    end for
end for
```

Slika 2.4. Viola-Jones algoritam raspoznavanja

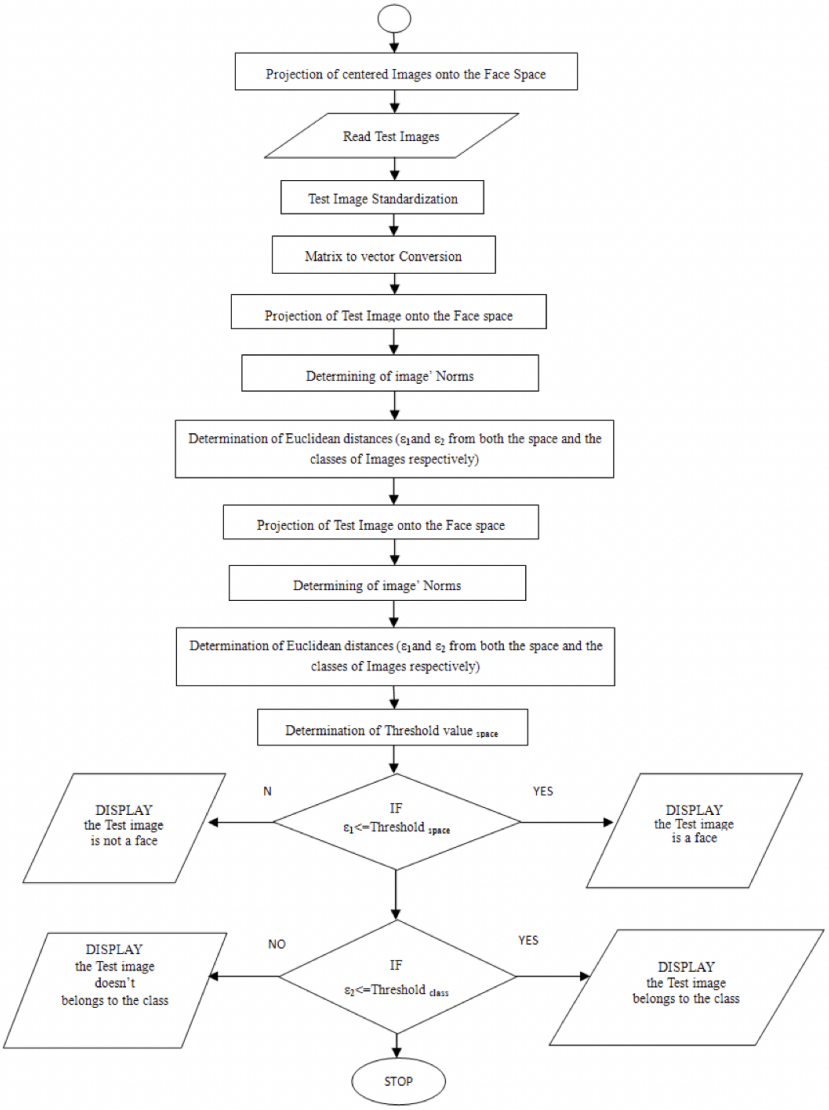
Glavna četiri koraka su izračunavanje Haarovih značajki (analiza susjednih pravokutnih područja, zbrajanje intenziteta piksela, izračun razlika), stvaranje integralne slike, Adaboost algoritam i kaskadni klasifikator. Nakon izračuna svih značajki, većina njih je beznačajna. Na slici 2.5. je prikaz dvije korisne značajke. Prva stavlja fokus na činjenicu da je područje gdje su oči često tamnije u usporedbi s nosom i obrazima. Druga je zasnovana na svojstvu da je područje između očiju svjetlije od područja očiju. Za izbor najboljih značajki svaka od njih se primjenjuje na uvježbanim slikama. One s najmanjom stopom pogreške su ujedno i najbolje. Primjena navedenih značajki na bilo kojem području na slici osim očiju nema smisla. Iz tog razloga uvodi se kaskadni klasifikator. Značajke se grupiraju u faze i primjenjuju se pojedinačno. Područje na slici koje nije područje lica se odbacuje i više se ne provjerava.[14] Ovaj algoritam se pokazao kao prilično efikasan u realnom vremenu i dugo je vremena bio korišten od strane Applea. [15]



Slika 2.5. Primjer značajki

Eigenfaces

Ovaj algoritam je nastao 1987. godine, a osmislili su ga Matthew Turk i Alex Pentland. Ideja algoritma leži u značajci svojstvenog lica (engl. *eigenfaces*) predstavljenog svojstvenim vektorima. Za kreiranje niza vektora potreban je veliki skup podataka, odnosno slika lica. Jako je važno da su sve slike iste svjetline, kao i veličine. Osim navedenog, oči i usne na svim uzorcima moraju biti poravnati. Algoritam ocjenjuje područja lica koja su svjetlija. [13] U odnosu na ostale algoritme, oni koji su bazirani na osvjetljenju su mnogo brži i efikasniji, ali im je nedostatak potreba za jednakim osvjetljenjem i poravnanjem. Ovo je jedan od najpopularnijih algoritama.



Slika 2.6. Dijagram tijeka Eigenfaces algoritma

Ostale metode

Od ostalih metoda, jedna od prvih ikad je Nixonova metoda. Temelji se na izračunu razmaka između očiju, a promatra se gradijent. Jer su u to vrijeme fotografije bile crno-bijele, promatrala se nijansa sive. Tu je još i Fisherfaces, koji je uz Eigenfaces jedan od popularnijih algoritama. Nadovezuju se jedan na drugi, ali se kod Fisherfaces analiza bazira na svođenje višedimenzionalnog vektora značajki na jednu dimenziju.

3. TEHNOLOGIJE KORIŠTENE ZA RAZVOJ APLIKACIJE

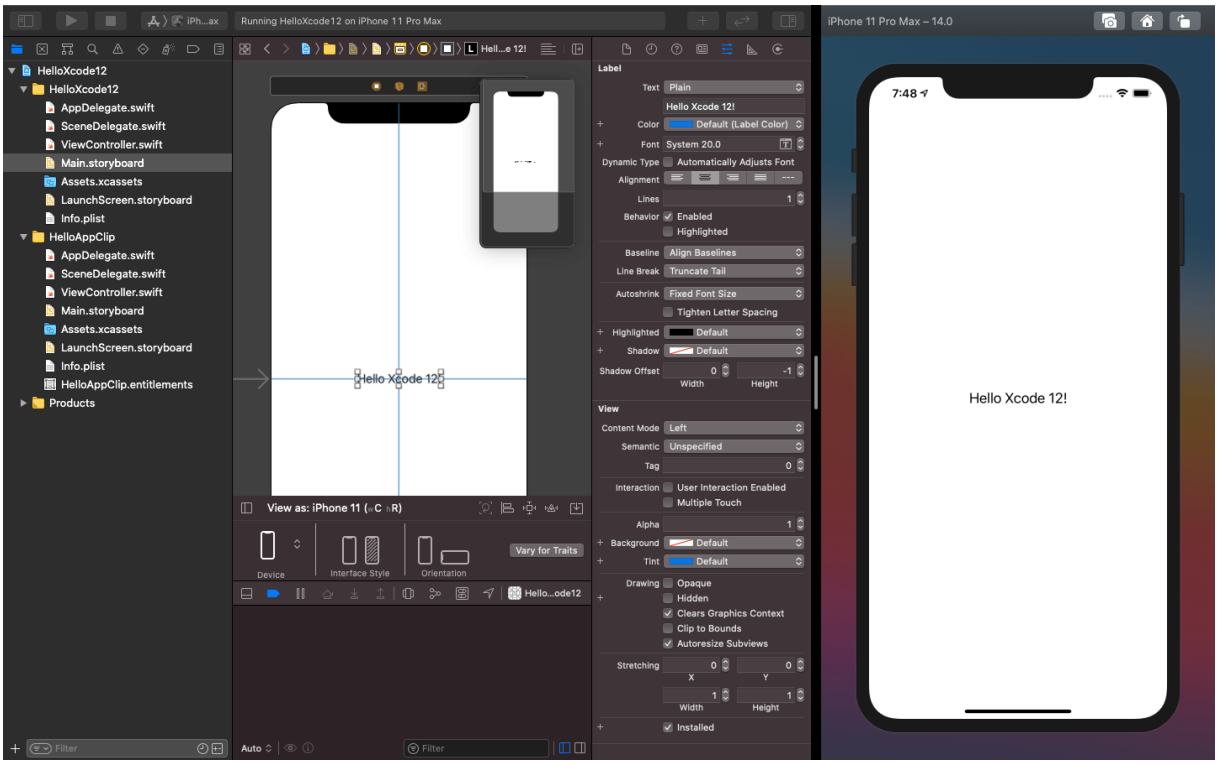
Apple ima svoj zatvoreni eko sustav i ne dopušta velika odstupanja prilikom izbora tehnologija izrade aplikacija, te su iz toga razloga za gotovo sve dijelove aplikacije korišteni službeno predviđeni alati. Korišteno je razvojno okruženje Xcode, programski jezik Swift, biblioteka UIKit, te Vision biblioteka za detekciju lica. Sve navedeno je objašnjeno u nastavku.

3.1. iOS aplikacije i operacijski sustav iOS

Operacijski sustav koji pogoni iPhone naziva se iOS, te postoji više verzija tog sustava, a u trenutku pisanja ovog rada posljednja verzija je iOS 15. Od uređaja koji su izdani u posljednje 4 godine 90% ih koristi iOS 14, dok čak 85 % svih uređaja ikad koristi iOS 14. [4] Zbog tako velikog postotka je iOS 14 izabran kao minimalna verzija sustava koju uređaj mora imati za rad aplikacije. Za objavu aplikacija na App Store potrebno je godišnje plaćati 100$, a u sklopu toga Apple se brine za naplatu i distribuciju aplikacije korisnicima. Aplikacija mora poštivati pravila koja je Apple definirao, te svaka aplikacija ide na provjeru. Iz navedenih razloga aplikacija trenutno nije objavljena na App store. [5]

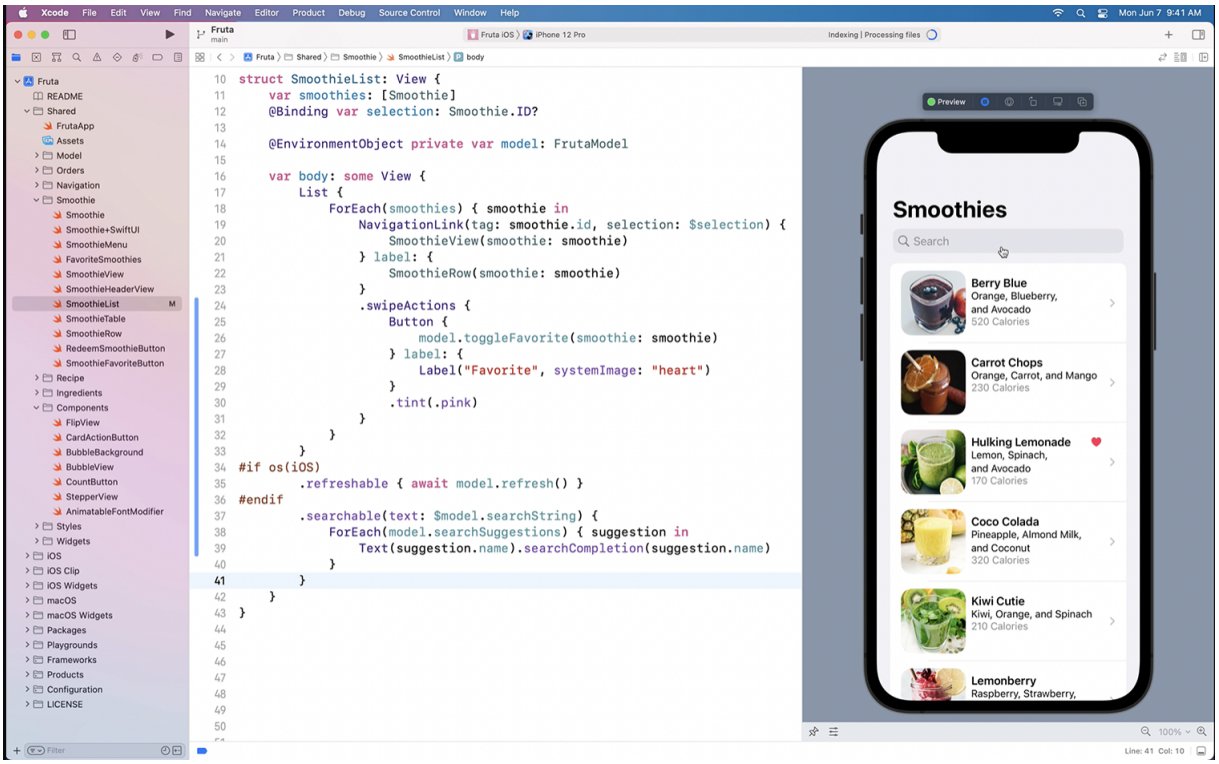
3.2. Xcode, Swift, UIKit

Xcode je Appleovo integrirano programsko razvojno okruženje za razvijanje aplikacija za sve Appleove proizvode. Tu spadaju iPhone, iPad, Apple TV, Apple Watch, kao i prijenosna i stolna računala. Trenutno je posljednja verzija 12.5.1. Za pokretanje Xcodea potrebno je imati računalo s macOS operacijskim sustavom. Testiranje je moguće obaviti na fizičkom uređaju ili na simulatoru. Simulator ima neka ograničenja, a budući da se u ovom primjeru koristi kamera, potrebno je koristiti fizički uređaj. Izgled Xcode programskog sučelja prikazan je na slici ispod.



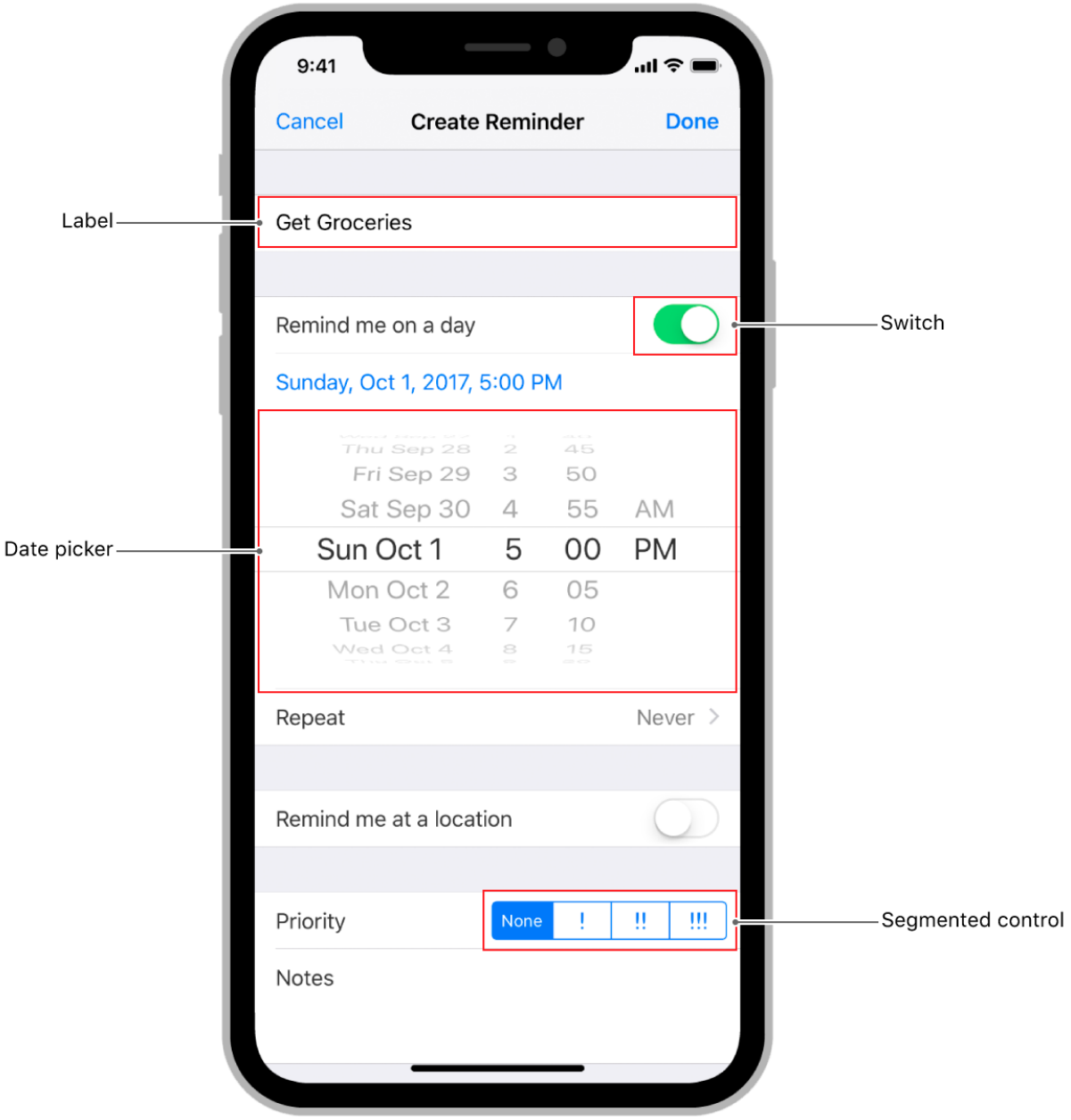
Slika 3.1. Razvojno okruženje Xcode

Swift je relativno novi (2014.) programski jezik koji je trenutno standard za pisanje aplikacija za iOS. Naslijedio je Objective-C, zbog toga što je jednostavniji, brži i sigurniji. I dalje je dozvoljena upotreba Objective-C za posebne zadatke, ali je za veliku većinu problema Swift samostalno dovoljan. Uvodi neke novitete poput Optional tipova, automatskog brojanja referenci (engl. *automatic reference counting*), uređenih parova i drugih značajki koje se javljaju u suvremenim programskim jezicima. Mnogo je sigurniji za pisanje jer obavlja puno više provjera prilikom prevođenja nego Objective-C. Omogućuje koncept proširivosti na skoro sve elemente, od klasa, preko tipova podataka do protokola. Apple promiče ovu promjenu paradigmi programiranja te ju naziva „protokolno orijentirano programiranje“. Primjer korisničkog sučelja napisanog koristeći SwiftUI je vidljiv na slici 3.2. u nastavku. Swift kao programski jezik trenutno je u verziji 5, a moguće ga je vježbati i online, kao i na drugim sustavima.



Slika 3.2. SwiftUI

UIKit je biblioteka koji se koristi za izgradnju korisničkog sučelja. Omogućuje traženu infrastrukturu za aplikaciju, arhitekturu prozora i pogleda, rukovanje događajima, višestrukim dodirima kao i ostalim načinima unosa podataka u aplikaciju. Podržava animacije, dokumente, crtanje i ispis, informacije o trenutnom uređaju, prikaz i upravljanje tekstom, upravljanje resursima, pristupačnost, pretraživanje i još mnogo toga. Apple trenutno razvija novi način pisanja korisničkog sučelja upotrebom SwiftUI-a, međutim u trenutnoj verziji još nije produkcijski spreman, te je za izradu ove aplikacije korišten UIKit. Izrada korisničkog sučelja korištenjem UIKita je moguća na dva načina: upotrebom *storyboarda* i programatski. *Storyboard* omogućuje drag and drop elemenata te njihovo jednostavno postavljanje s trenutnim pregledom promjena. Međutim za postizanje određenih efekata ili izgleda potrebno je dodati malo koda. Ako se koristi programatski način, sve se piše u kodu. Nema trenutnog prikaza promjena te je za prikaz promjena potrebno svaki put pokrenuti aplikaciju. *Storyboard* je pristupačniji za početnike jer je lakše dobiti pregled nad cijelom aplikacijom. Prilikom izrade aplikacije korišten je storyboard, uz dodavanje koda po potrebi. Na slici 3.3. su prikazani neki najosnovniji elementi i njihovi nazivi.



Slika 3.3. Elementi korisničkog sučelja

3.3. Vision biblioteka

S dolaskom iOS verzije 11 Apple je predstavio svoj programski okvir pod nazivom Vision. Primjenjuje algoritme za računalni vid za izvođenje različitih zadataka na ulaznim slikama odnosno videozapisima. Vision programski okvir se može koristiti za detekciju lica i značajki lica, detekciju teksta, prepoznavanje bar kodova, te praćenje značajki. [6] Također omogućuje

korištenje vlastitih Core ML modela za zadatke poput klasifikacije ili detekcije objekata. Unutar Visiona postoje 3 uloge:

1) Zahtjev (engl. *request*) - koristi se kada se od programskog okvira zatraži prepoznavanje nečega, te opisuje zahtjev. Postoji više mogućih zahtjeva poput *VNDetectBarcodesRequest* za prepoznavanje bar kodova, *VNDetectTextRectanglesRequest* za detekciju područja teksta. Unutar ovog projekta su korišteni *VNDetectFaceRectanglesRequest* za detekciju pravokutnika oko lica koje je prepoznato. Osim navedenog koristi se i zahtjev *VNDetectFaceLandmarksRequest* za dohvaćanje značajki lica. Detaljnije će biti objašnjeni u nekom od narednih poglavlja.
2) Rukovatelj zahtjevom (engl. *request handler*) - izvršava zahtjev koji mu je predan. Koristi se kada se želi da programski okvir nešto uradi, procesuira jedan ili više zahtjeva. Postoje dva tipa: *VNImageRequestHandler* koji analizira sliku i *VNSequenceRequestHandler* koji analizira niz slika, te se najčešće koristi za praćenje objekta.
3) Zapažanja (engl. *observations*) - rezultati zahtjeva zatraženog od programskog okvira su zapakirani unutar zapažanja. Zapažanje se naziva svaka detekcija, pa ako se radi npr. o jednoj faci postoji samo jedno zapažanje. Rezultat zapažanja je potrebno prikazati kao jedan od tipova zapažanja, a unutar ovog projekta koristi se *VNFaceObservation*.

3.4. MVC pristup dizajnu programske podrške

MVC (engl. *model-view-controller*) je dizajn obrazac za kreiranje aplikacija s čestom primjenom prilikom izrade iOS aplikacija. Odgovor je na pitanje kako organizirati kod. Zasnovan je na objektno orijentiranoj paradigmi. Struktuira tok podataka i interakciju s aplikacijom, te opisuje kako bi se podatci trebali prebaciti s jednog dijela aplikacije na drugi. Sastoji se od 3 komponente: pogleda (engl. *view*), modela (engl. *model*) i upravitelja (*engl. controller*). Model enkapsulira podatke, te sadrži logiku za upravljanje tim podatcima. Pogled je objekt kojeg korisnik može vidjeti unutar korisničkog sučelja. Upravitelj sadrži logiku koja se brine za sve što ide između pogleda i modela. Omogućuje komunikaciju od pogleda prema modelu i obrnuto. Elementi ove aplikacije su rađeni prema ovom obrascu.

3.5. Firebase baza podataka u stvarnom vremenu

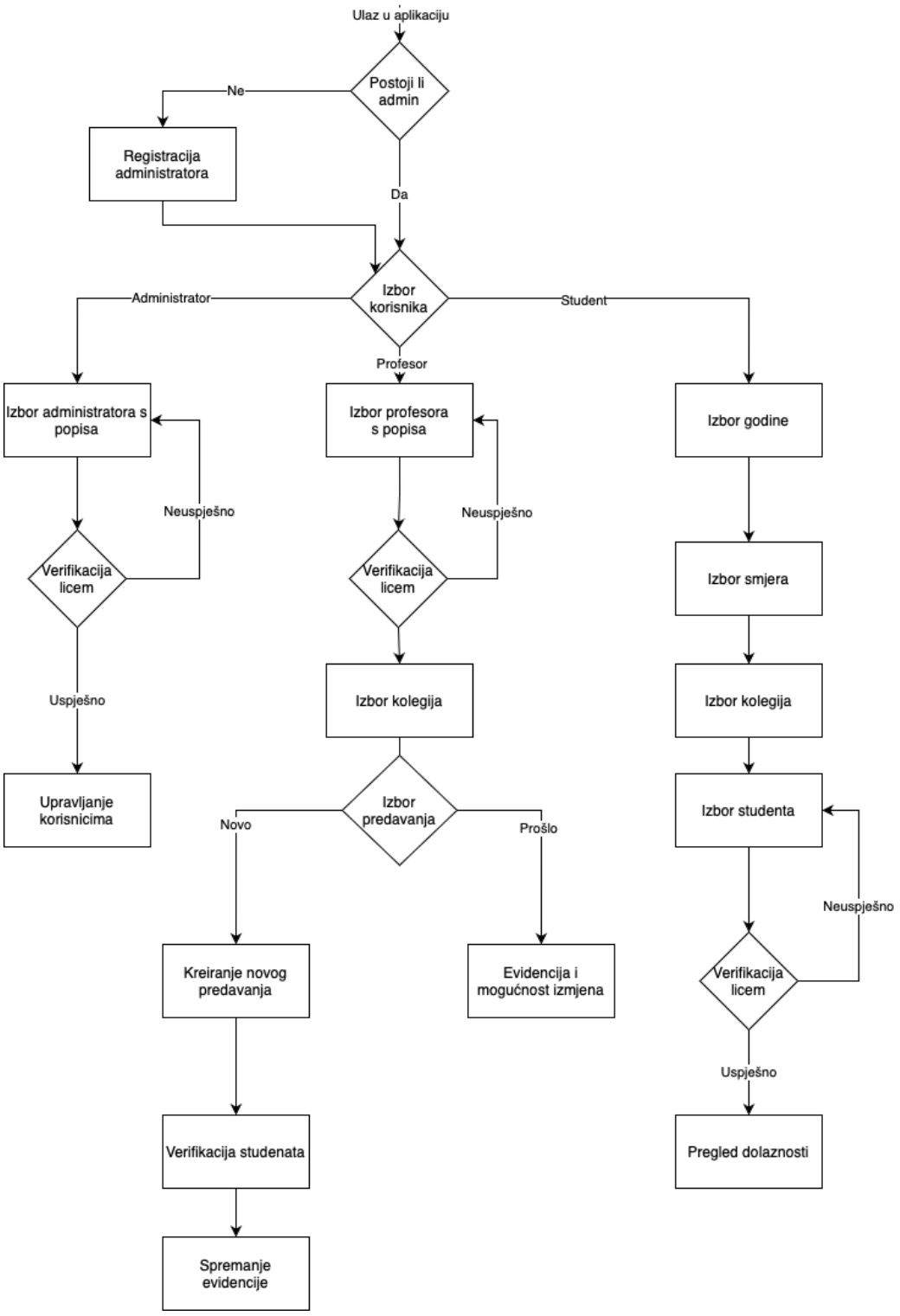
Za potrebe spremanja podataka korištena je Googleova platforma koja pruža pozadinsku podršku kao uslugu (engl. *backend as a service*) pod nazivom Firebase. Inače se koristi kao pomoć pri razvoju internetskih i mobilnih aplikacija. Sve usluge koje Firebase nudi nalaze se u oblaku računala te im se pristupa putem aplikacijskog sučelja. Osim pohrane podataka u stvarnom vremenu sadrži još brojne mogućnosti poput osiguravanja potrebne logike za registraciju i prijavu korisnika, pristupa udaljenoj memoriji, posluživanja, testiranja programskih rješenja, kao i alata za praćenje rada aplikacije i mjerenja učinkovitosti. Međutim, za potrebu diplomskog rada, bilo je potreba samo za bazom podataka u stvarnom vremenu.

4. APLIKACIJA ZA PRIJAVU PUTEM PREPOZNAVANJA LICA

Glavne odlike programskog rješenja su registracija i prijava, prepoznavanje značajki lica, spremanje i evidencija dolaznosti. Unutar ovog poglavlja su prikazane osnovne komponente od kojih je sačinjena ova aplikacija. Uz svaku komponentu dan je prikaz zaslona koji je specifičan za nju. Aplikaciju je potrebno instalirati na fizički uređaj s minimalnom verzijom sustava iOS 14, te joj prilikom upita dozvoliti upotrebu kamere. Ideja aplikacije je vođenje evidencije dolaznosti studenata na nastavu putem prepoznavanja lica. Aplikacija je rađena prema modelu izvođenja nastave na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku. Postoje tri tipa korisnika: administrator, profesor i student. Aplikacija je zamišljena tako da jednom kada su podatci u njoj popunjeni, što je objašnjeno u nastavku, profesor dolazi na nastavu i prijavljuje se u aplikaciju upotrebom svog lica. Nakon što se prijavi, bira kolegij za koji drži predavanje i otvara novo predavanje. Tada predaje mobitel studentima koji odabiru sebe i potvrđuju svoj identitet. Student nema drugih mogućnosti prilikom tog dijela, a kada zadnji student završi s verifikacijom vraća mobitel profesoru koji jedini može spremiti evidenciju.

4.1. Dijagram tijeka aplikacije

Na slici 4.1. je prikazan dijagram tijeka aplikacije koji pokazuje navigaciju kroz elemente aplikacije. U narednim potpoglavljima su navedeni elementi pojedinačno objašnjeni.



Slika 4.1. Dijagram tijeka aplikacije

4.2. Registracija

Za prijavu i registraciju je potrebno koristiti nešto jedinstveno za svakog korisnika. U ovom slučaju korištena je ljudska faca kao dovoljno unikatna. Ovdje postoji problem što dvije

osobe mogu biti dosta slične ili čak blizanci. Međutim prema izvorima [7] na svijetu je samo 2.3% blizanaca, a budući da je svrha ove aplikacije upotreba unutar grupa studenata na fakultetu, postotak blizanaca koji idu na isti smjer na fakultetu je daleko manji, te je unutar ove aplikacije zanemaren. Moguće poboljšanje u nekoj od narednih verzija aplikacije bi bilo uvođenje oznake za blizance, te bi profesor njih morao ručno provjeriti. I dalje je upitno koliko profesor može biti siguran u svoju odluku.

4.2.1. Registracija administratora

Prilikom prvog pokretanja aplikacije traži se unos podataka o administratoru, odnosno odgovornoj osobi koja će biti zadužena za dodavanje ili uklanjanje profesora, kao i izmjenu podataka o njima. Administrator bi trebao biti odgovorna osoba iz IKT sektora, budući da će imati pravo punog pristupa svim podatcima. Administrator može biti i neki od profesora, te to neće utjecati na njegov profesorski profil. Za registraciju administrator mora unijeti svoje ime, prezime, te dozvoliti uzimanje i spremanje značajki svog lica, a taj dio je objašnjen u poglavlju 5. Nakon što se administrator registrira, njegovi podatci se spremaju i otvara se novi zaslon za prijavu. Taj administrator ostaje spremljen za daljnju upotrebu aplikacije, te ga je moguće promijeniti unutar postavki aplikacije. Za promjenu administratora potrebna je njegova verifikacija licem, tako da trenutni i novi administrator moraju biti zajedno prilikom postavljanja novog administratora.

4.2.2. Registracija profesora

Profesora može registrirati samo administrator, te administrator potvrđuje identitet profesora. Administrator i profesor moraju biti skupa prilikom registracije profesora budući da administrator mora svojim licem potvrditi svoj identitet, te dodati profesorovo lice za profil profesora. Od profesora se još traži ime, prezime, elektronička pošta i njegova titula.

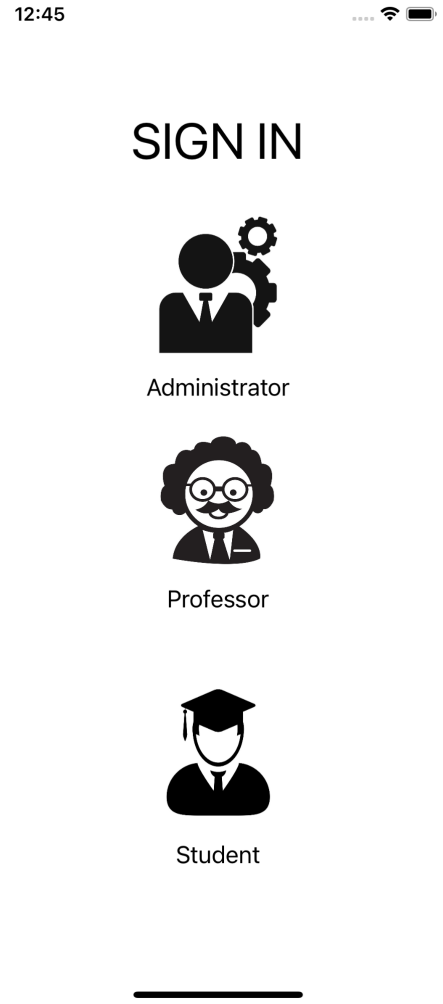
4.2.3. Registracija studenta

Za registraciju studenata ovlašteni su i administrator i profesori. Profesor može dodati studente na prvom predavanju ili na nekom od budućih, ali u idealnom slučaju studenti bi bili dodani prilikom slikanja za studentsku iskaznicu. Od studenata se traže ime, prezime i odabir kojoj godini i smjeru pripadaju. Budući da se studenti mijenjaju tijekom studija, nekada će biti

potrebno ponovno snimiti značajke lica za nekog studenta. Taj student mora kontaktirati nekog profesora ili administratora.

4.3. Prijava i mogućnosti

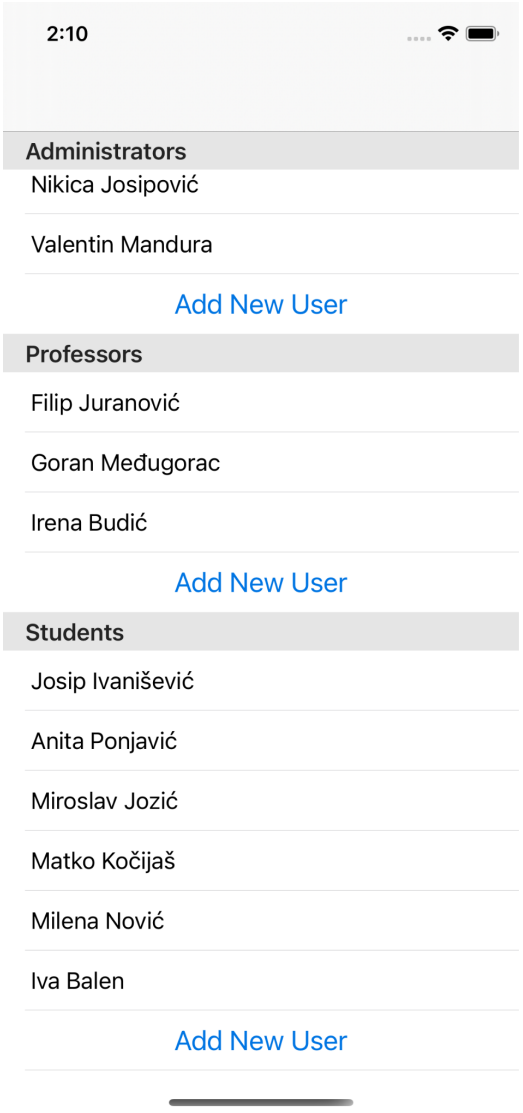
Nakon registracije, pri svakom sljedećem ulasku u aplikaciju se pojavljuje zaslon na kojemu je potrebno odabrati tip korisnika koji se želi prijaviti. U ponudi su administrator, profesor i student. Nakon odabira bilo kojeg od ponuđenih otvara se lista korisnika koji su spremljeni. Korisnik treba izabrati sebe, nakon čega mu se otvara prozor za verifikaciju lica. Nakon što je identitet korisnika potvrđen, otvara se novi zaslon s mogućnostima koje su dostupne za tog korisnika.



Slika 4.1. Izgled početnog zaslona

4.3.1. Prijava administratora

Ako se na početnom zasloni izabere administrator pristupa se popisu ovlaštenih administratora. Može ih biti jedan ili više. Administrator posjeduje prava punog pristupa, a njegove mogućnosti su: dodavanje novog profesora, dodavanje novog studenta, brisanje svih tipova korisnika kao i izmjena podataka o svakom korisniku. Osim navedenog što će administrator često koristiti moguće je i dodati novog administratora. Ako netko ne želi više biti administrator, mora dodijeliti ta prava nekome drugome i taj drugi administrator ga mora ukloniti.



Slika 4.2. Korisničko sučelje prijave administratora

4.3.2. Prijava profesora

Profesor, nakon što na početnom zaslonu odabere prijavu za profesore, bira svoje ime s popisa i vrši verifikaciju licem. Jednom kada se prijavi ima pregled svojih kolegija kao i mogućnost dodavanja novog kolegija. Nakon što odabere željeni kolegij vidi datume prošlih predavanja, gdje odabirom određenog datuma može vidjeti za svakog studenta dolaznost. Također moguće je promijeniti dolaznost za određenog studenta. Dodavanjem novog predavanja, profesor prepušta mobitel studentima koji se po tome prijavljuju. Nakon što se prijave, profesor sprema dolaznost klikom na gumb na dnu i ponovno svojim licem potvrđuje željenu radnju, u svrhu sprečavanja slučajnog zaključavanja od strane nekog studenta.

Slika 4.3. Prikaz kolegija profesora

4.3.3. Prijava studenta

Student prilikom prijave na početnom zaslonu, bira svoju godinu, smjer i kolegij za koji može provjeriti svoju dolaznost. Student nema nikakve mogućnosti uređivanja zbog sprječavanja

krivotvorenja značajki lica više studenata s jednim. Iz tog razloga ako aplikacija ne prepoznaje studenta, potrebno je kontaktirati profesora ili administratora koji će utvrditi identitet.

4.4. Značajke lica

Prilikom registracije ili prijave bilo kojeg tipa korisnika potrebno je identitet potvrditi licem. To se radi tako da se značajke lica koje su spremljene prilikom registracije usporede sa značajkama lica koja aplikacija trenutno prepozna. Taj modalni prozor za verifikaciju se otvara i samo javlja je li to taj korisnik ili ne.

4.5. Evidencija dolaznosti

Ako su prijavljeni profesor ili student mogu vidjeti dolaznost za svoje kolegije. Profesor kada se verificira može provjeriti dolaznost za sve studente, dok student koji je verificiran može vidjeti samo svoju dolaznost. Student može samo pregledati dolaznost i ako ima neku zamjerku, mora kontaktirati profesora koji može mijenjati podatke, kako bi mu promijenio dolaznost za navedeni datum.

5. PROGRAMSKO RJEŠENJE APLIKACIJE

Unutar navedenog poglavlja osnovne ideje i dijelovi aplikacije su objašnjeni i potkrijepljeni programskim kodom.

5.1. Korisničko sučelje

U nastavku su objašnjeni osnovni elementi koji su korišteni za izradu korisničkog sučelja.

5.1.1. UITableView

Korisničko sučelje izrađeno je koristeći storyboard za sve zaslone aplikacije, osim zaslona verifikacije i uzimanja značajki lica korisnika koje je napisano unutar koda. Za prikaz listi profesora, administratora, studenata, kolegija, kao i dolaznosti je korišten element *UITableView*. To je element koji omogućuje prikaz elemenata u redovima jednog ispod drugog. Omogućuje skrolanje, ponovno iskorištavanje istih ćelija i efikasno iskorištavanje memorije. [8] Na slici 5.1. se vidi implementacija prikaza svih kolegija određenog profesora.

```
func tableView(_ tableView: UITableView, titleForHeaderInSection
    section: Int) -> String? {
    switch section {
    case 0:
        return "Administrators"
    case 1:
        return "Professors"
    case 2:
        return "Students"
    default:
        return ""
    }
}

func tableView(_ tableView: UITableView, didSelectRowAt indexPath:
    IndexPath) {

    if indexPath.section == 0 && indexPath.row == admins.count {
        let nextScreen = "AddNewUser"
        let storyboard = UIStoryboard(name: nextScreen, bundle: nil)
        let nextViewController = storyboard
            .instantiateViewController(withIdentifier: String(nextScreen
                + "ViewController"))
        navigationController?.pushViewController(nextViewController,
            animated: true)
    }

    if indexPath.section == 1 && indexPath.row == professors.count {
        let nextScreen = "AddNewUser"
        let storyboard = UIStoryboard(name: nextScreen, bundle: nil)
        let nextViewController = storyboard
            .instantiateViewController(withIdentifier: String(nextScreen
                + "ViewController"))
        navigationController?.pushViewController(nextViewController,
            animated: true)
    }

    if indexPath.section == 2 && indexPath.row == students.count {
        let nextScreen = "AddNewUser"
        let storyboard = UIStoryboard(name: nextScreen, bundle: nil)
        let nextViewController = storyboard
            .instantiateViewController(withIdentifier: String(nextScreen
                + "ViewController"))
        navigationController?.pushViewController(nextViewController,
            animated: true)
    }
}
```

```
extension AdminLoggedViewController : UITableViewDataSource{

    func numberOfSections(in tableView: UITableView) -> Int {
        return 3
    }

    func tableView(_ tableView: UITableView, numberOfRowsInSection section:
        Int) -> Int {
        switch section {
        case 0:
            return admins.count + 1
        case 1:
            return professors.count + 1
        case 2:
            return students.count + 1
        default:
            return 1
        }
    }

    func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath)
        -> UITableViewCell {

        switch indexPath.section {
        case 0:
            if indexPath.row == admins.count {
                let cell = tableView.dequeueReusableCell(withIdentifier:
                    String(describing: AddNewUserTableViewCell.self), for:
                    indexPath) as! AddNewUserTableViewCell
                return cell
            }
            let cell = tableView.dequeueReusableCell(withIdentifier:
                String(describing: UserTableViewCell.self), for: indexPath) as!
                UserTableViewCell
            cell.configureLabel(with: admins[indexPath.row].name)
            return cell
        case 1:
            if indexPath.row == professors.count {
                let cell = tableView.dequeueReusableCell(withIdentifier:
                    String(describing: AddNewUserTableViewCell.self), for:
                    indexPath)  as! AddNewUserTableViewCell
                return cell
            }
            let cell = tableView.dequeueReusableCell(withIdentifier:
                String(describing: UserTableViewCell.self), for: indexPath) as!
                UserTableViewCell
            cell.configureLabel(with: professors[indexPath.row].name)
            return cell
```

Slika 5.1. Tablični prikaz svih korisnika

Prilikom popunjavanja tablice ćelijama je potrebno paziti da se popuni točan broj. Zbog toga se svi podatci nalaze u poljima, nad kojima funkcija count vraća broj elemenata. Nadalje, ako je riječ o posljednjoj ćeliji, stavlja se drugi tip ćelije, odnosno ćelija za dodavanje novog korisnika. Ponašanje ćelija prilikom ponovnog korištenja je definirano u zasebnim datotekama.

5.1.2. UINavigationController

Upravitelj navigacije omogućuje kretanje kroz aplikaciju i korišten je za osiguravanje fluidnog rada. Postoje dva načina prikaza novog zaslona aplikacije: *push* i *present*. ako se koristi push način, novi zaslon se dodaje na stog i koristi se kada se očekuje daljnja navigacija ili duže zadržavanje na određenom zaslonu. [9] *Push* način je korišten za navigaciju od početnog zaslona do krajnjih točaka, dok je present način iskorišten za verifikaciju lica korisnika i uzimanje značajki zato što se ta radnja odvija mali dio vremena i služi samo kao povratna informacija. Na slici 5.2. je prikazan dio koda koji prikazuje dvije funkcije. Primjer je dan za administratora, no slično je i za ostale korisnike. Kada se administrator želi prijaviti licem poziva se funkcija *navigateToCheckFace* kojoj se predaje korisnik u ovom slučaju administrator. Tu se instancira novi upravitelj prikaza kojem se predaju značajke lica administratora. Unutar zatvorenja koje se postavlja unutar njega, te ako značajke budu odgovarale se poziva funkcija successfullyLogged koja će prijaviti administratora i uputiti ga na novi zaslon. Ako upravitelj prepozna značajke sam se zatvara. Nakon što se određeni korisnik prijavi, postoji i mogućnost odjave, kako bi se prijavio neki drugi, a za navigaciju kroz sve navedeno zadužen je upravitelj navigacije.

```
extension AdminsViewController {
    func succesfullyLogged(user: Admin){

        let storyboard = UIStoryboard(name: "AdminLogged", bundle: nil)
        let nextViewController = storyboard
            .instantiateViewController(withIdentifier:  "AdminLoggedViewController") as!
                AdminLoggedViewController
        nextViewController.admin = user
        navigationController?.setViewControllers([nextViewController], animated: true)
}
}

private extension AdminsViewController {

    func navigateToCheckFace(for admin: Admin) {
        let storyboard = UIStoryboard(name: "CheckFace", bundle: nil)
        let checkFaceViewController = storyboard
            .instantiateViewController(withIdentifier: String(describing: CheckFaceViewController.self)
            ) as! CheckFaceViewController
        checkFaceViewController.features = admin.faceFeatures
        checkFaceViewController.nextScreenIfSuccess = "ProfessorLogged"

        checkFaceViewController.completion = {
            self.succesfullyLogged(user: admin)
        }
        navigationController?.present(checkFaceViewController, animated: true, completion: nil)
    }
}
```

Slika 5.2. Prikaz navigacije upotrebom Navigation upravitelja

5.1.3. CheckFaceViewController

Kada se dogodi potreba za uzimanjem značajki lica, privremeno se prikazuje modalni zaslon. Unutar ovog potpoglavlja je objašnjen samo dio koji se odnosi na prikaz elemenata na zaslonu. Prilikom učitavanja zaslona prva funkcija koja se poziva je *setupCamera*() koja osigurava da se radi o uređaju koji posjeduje fizičku kameru. Iz tog razloga navedeno nije moguće testirati na simulatoru, nego je potreban fizički uređaj. Koristi se prednja kamera, a ulaz s nje se koristi u nastavku. Nakon toga se postavlja pretpregled unutar *setPreview()* funkcije gdje se između ostalog postavljaju postavke za video i omjer tog videa. Budući da se ovdje događaju provjere je li lice unutar ekrana i prepoznavanje značajki lica, dodaje se još jedan sloj koji prikazuje pravokutnik oko lica i crte koje prikazuju prepoznate elemente lica. Pravokutnik je obojen zelenom bojom, a značajke crvenom. Po potrebi se pojavljuje tekst bijele boje s uputama za korisnika. ako je prepoznavanje lica uspješno, zaslon se sam zatvara uz prigodnu animaciju.

```swift
private func setupCamera() {
    let deviceDiscoverySession = AVCaptureDevice.DiscoverySession(deviceTypes:
        [.builtInWideAngleCamera], mediaType: .video, position: .front)
    guard let device = deviceDiscoverySession.devices.first,
          let deviceInput = try? AVCaptureDeviceInput(device: device),
          captureSession.canAddInput(deviceInput)
    else { return }

    captureSession.addInput(deviceInput)
    setupPreview()
}

private func setupPreview() {
    previewLayer.videoGravity = .resizeAspectFill

    cameraView.layer.addSublayer(previewLayer)
    previewLayer.frame = cameraView.layer.bounds

    videoDataOutput.videoSettings = [(kCVPixelBufferPixelFormatTypeKey as NSString) :
        NSNumber(value: kCVPixelFormatType_32BGRA)] as [String : Any]
    videoDataOutput.setSampleBufferDelegate(self, queue: DispatchQueue(label: "camera queue"))

    captureSession.addOutput(videoDataOutput)

    let videoConnection = videoDataOutput.connection(with: .video)
    videoConnection?.videoOrientation = .portrait

}
```

Slika 5.3. Prikaz koda

5.2. DETEKCIJA LICA I ZNAČAJKI LICA

Unutar ovog poglavlja objašnjeno je prepoznavanje dijela slike na kojem se nalazi lice, te izvlačenje značajki s tog lica. Budući da je videozapis sastavljen od određenog broja sličica u sekundi, zapravo obrada videozapisa se vrši tako da se promatra svaka sličica pojedinačno. Prvo što je bitno osigurati je da slika postoji, te se to vidi u drugoj liniji koda funkcije sa slike ispod. ako ulaz ne postoji, detekcija se bespotrebno neće ni izvršavati.

5.2.1. Zahtjev i rukovatelj zahtjevom

Nakon što se utvrdi da slika postoji kreira se zahtjev za prepoznavanje lica. On je tipa *VNDetectFaceLandmarksRequest*. Nakon toga kreira se rukovatelj zahtjevom koji kao parametre prima sliku, oznaku za zrcaljenje lijevo, te se za dodatne mogućnosti predaje prazan rječnik. Rukovatelj zahtjevom izvodi zahtjev, te u slučaju greške hvata pogrešku. Unutar zahtjeva je postavljeno da nakon što se uspješno izvrši osigura da je rezultat tipa *VNFaceObservation*, odnosno prekida izvođenje u suprotnom. Rezultat se naziva zapažanja, te nakon što se utvrdi da

su zapažanja odgovarajućeg tipa poziva se funkcija *handleFaceDetectionObservations* koja služi kao rukovatelj navedenim zapažanjima.

```
func captureOutput(_ output: AVCaptureOutput, didOutput sampleBuffer: CMSampleBuffer, from connection:
    AVCaptureConnection) {
    print("CameraVC: ", sampleBuffer)
    guard let imageBuffer = CMSampleBufferGetImageBuffer(sampleBuffer) else {return}

    let faceDetectionRequest = VNDetectFaceLandmarksRequest(completionHandler: { (request: VNRequest,
        error: Error?) in

        DispatchQueue.main.async {
            self.faceLayers.forEach { drawing in
                drawing.removeFromSuperlayer()
            }

            guard let observations = request.results as? [VNFaceObservation] else {return}
            self.handleFaceDetectionObservations(for: observations)
        }

    })

    let imageRequestHandler = VNImageRequestHandler(cvPixelBuffer: imageBuffer, orientation:
        .leftMirrored, options: [:])
    do{
        try imageRequestHandler.perform([faceDetectionRequest])
    } catch {
        print("Error in CameraVC:", error.localizedDescription)
    }
}
```

Slika 5.4. Kod funkcije *captureOutput*

5.2.2. Zapažanja i rukovatelj zapažanjima

Rukovatelj zapažanjima je napisan tako da prima polje zapažanja. Za svako zapažanje izvodi radnje prikazane na slici 5.5. Svako zapažanje sadrži podatak nazvan *boundingBox*. To su koordinate 4 točke, koje označavaju pravokutnik unutar kojeg se zapažanje tj. lice nalazi. Taj pravokutnik postoji samo ako je aplikacija uspješno prepoznala lice. Međutim navedeni kao takav može postojati, ali da njegove stranice prelaze granice na zaslonu koje su vidljive korisniku. Zbog toga se provjerava je li cijeli pravokutnik vidljiv na ekranu i ako nije ispisuje se poruka korisniku. S tim se postiže da je lice na sredini zaslona i da su sve značajke koje su bitne vidljive. Obrub pravokutnika se postavlja da bude zelene boje, dok se njegova unutrašnjost ostavlja prozirna. Nakon toga se provjerava postoje li značajke lica, te ako postoje značajke, provjerava se svaka posebno o čemu više u nastavku.

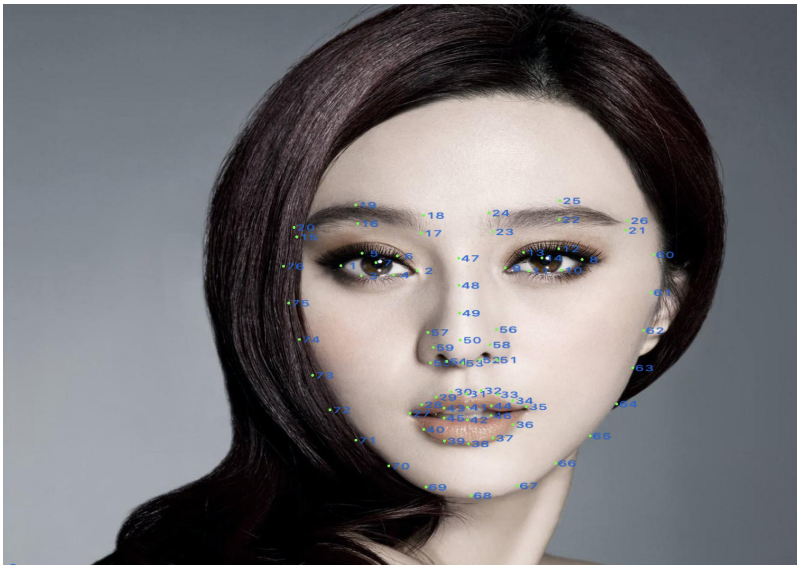
```swift
private func handleFaceDetectionObservations(for observations: [VNFaceObservation]) {
    observations.forEach { observation in
        let faceRectConverted =
            previewLayer.layerRectConverted(fromMetadataOutputRect:
            observation.boundingBox)
        let faceRectanglePath = CGPath(rect: faceRectConverted, transform: nil)

        guard isFaceInView(for: faceRectanglePath) else { return }

        let faceLayer = CAShapeLayer()
        faceLayer.path = faceRectanglePath
        faceLayer.fillColor = UIColor.clear.cgColor
        faceLayer.strokeColor = UIColor.green.cgColor

        faceLayers.append(faceLayer)
        containterView.layer.addSublayer(faceLayer)

        if let landmarks = observation.landmarks {

            if let faceContour = landmarks.faceContour {
                handleFaceContourLandmark(faceContour, faceBoundingBox:
                    faceRectConverted)
            }

            if let leftEye = landmarks.leftEye {
                handleLeftEyeLandmark(leftEye, faceBoundingBox: faceRectConverted)
            }
            if let leftEyebrow = landmarks.leftEyebrow {
                handleleftEyebrowLandmark(leftEyebrow, faceBoundingBox:
                    faceRectConverted)
            }
            if let rightEye = landmarks.rightEye {
                handleRightEyeLandmark(rightEye, faceBoundingBox: faceRectConverted)
            }
            if let rightEyebrow = landmarks.rightEyebrow {
                handleRightEyebrowLandmark(rightEyebrow, faceBoundingBox:
                    faceRectConverted)
            }

            if let nose = landmarks.nose {
                handleNoseLandmark(nose, faceBoundingBox: faceRectConverted)
            }

            if let outerLips = landmarks.outerLips {
                handleOuterLipsLandmark(outerLips, faceBoundingBox:
```

```swift
    private func isFaceInView(for faceRectangle: CGPath) -> Bool {

        if faceRectangle.boundingBox.origin.x > 20
            && faceRectangle.boundingBox.origin.y > 40
            && (faceRectangle.boundingBox.origin.x + faceRectangle.boundingBox.width) <
                (containterView.frame.width - 20)
            && (faceRectangle.boundingBox.origin.y + faceRectangle.boundingBox.height) <
                (containterView.frame.height)
        {
            return true
        }
        return false
    }

}
```

Slika 5.5. Rukovatelj zapažanjima

5.2.3. Značajke lica i rukovatelj značajkama

Vision biblioteka omogućuje detekciju ključnih značajki lica. Značajke lica su zapravo koordinate specifičnih točaka na licu. Ukupno na licu prepoznaje 76 točki koje su ključne za bilo kakvu detekciju, a te je točke najlakše objasniti primjerom na slici 5.6. Značajke cijelog lica se mogu podijeliti u skupine pa tako postoje značajke: lijevog oka, desnog oka, lijeve obrve, desne obrve, nosa, unutarnjih i vanjskih usana, te konture lica.



Slika 5.6. Raspored značajki na licu

Svaka skupina značajki ako postoji se predaje svom rukovatelju značajkama. Nisu bitne sve značajke lica nego se iz njih izvlače neki drugi podatci poput širine očiju, širine usana, visine nosa i druge značajke koje su navedene u poglavlju 5.2.4 Pomoću ovih značajki moguće je zaključiti još neke poput na primjer razmaka između obrva. Osim značajki rukovatelju se predaje i okvir oko lica koje je prepoznato. Koordinate točki se razlikuju u ovisnosti od uređaja, budući da govore gdje se na zaslonu točka nalazi. Zasloni nemaju iste širine ni visine, pa se zbog toga rukovatelju i predaje okvir oko lica. Na slici 5.7. je prikazan rukovatelj značajkama lijevog oka. Kada je pozvan, normalizira točke iz područja značajki u točke u koordinatnom sustavu, te za svaku točku pravi CGPoint s x i y vrijednosti. Skalira navedeno na položaj unutar okvira. Brine se za iscrtavanje i bojanje značajke, a na kraju se normalizirane točke predaju funkciji koja će obaviti potreban izračun i spremiti značajku.

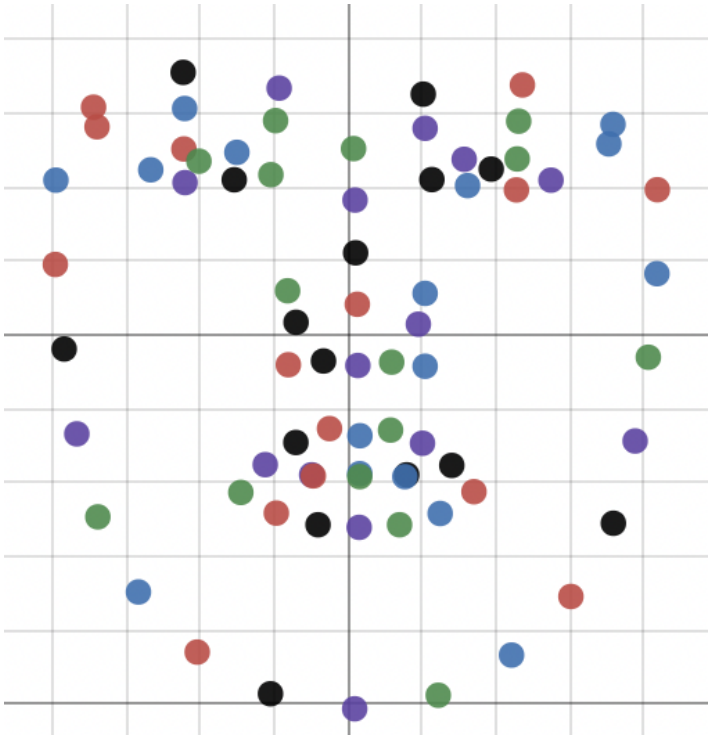
```
private func handleLeftEyeLandmark(_ eye: VNFaceLandmarkRegion2D, faceBoundingBox: CGRect) {
        let landmarkPath = CGMutablePath()
        let landmarkPathPoints = eye.normalizedPoints
            .map({ eyePoint in
                CGPoint(
                    x: eyePoint.y * faceBoundingBox.height + faceBoundingBox.origin.x,
                    y: eyePoint.x * faceBoundingBox.width + faceBoundingBox.origin.y)
            })

        landmarkPath.addLines(between: landmarkPathPoints)
        landmarkPath.closeSubpath()
        let landmarkLayer = CAShapeLayer()
        landmarkLayer.path = landmarkPath
        landmarkLayer.fillColor = UIColor.clear.cgColor
        landmarkLayer.strokeColor = UIColor.red.cgColor

        self.faceLayers.append(landmarkLayer)
        self.containterView.layer.addSublayer(landmarkLayer)
        self.saveLeftEye(leftEyePoints: landmarkPathPoints, boundingBox: faceBoundingBox)
}
```

Slika 5.7. Rukovatelj značajkama

5.2.4. Obrada značajki lica

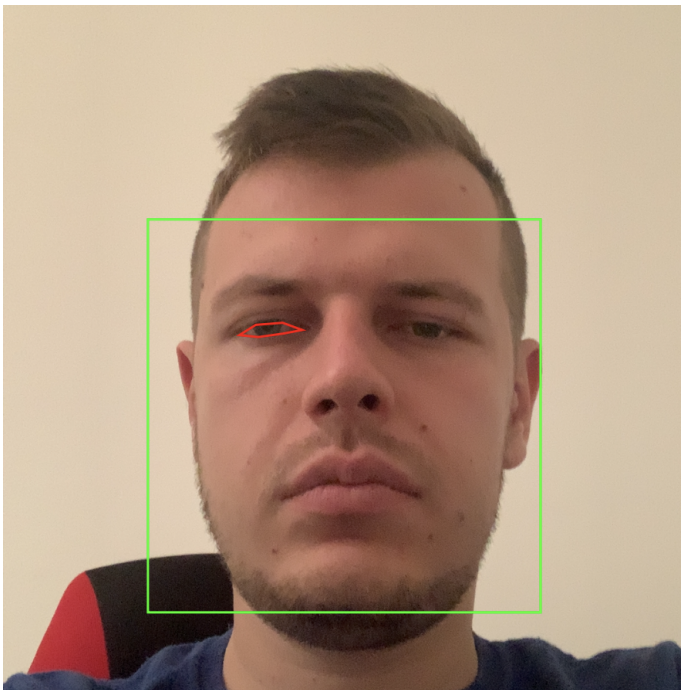
Za potrebe diplomskog rada su ispisane koordinate svih 76 značajki, gdje se dobilo 76 uređenih parova s vrijednostima X i Y koordinata značajke. Primjer izgleda značajki je: (0.2761819660663605, 0.2330799549818039), (0.2827988862991333, 0.39574018120765686) (0.2934877872467041, 0.2794997990131378), (0.3425232523452334, 0.532534523454553223)

(0.28967031836509705, 0.34614089131355286)...., te je vidljivo da je navedeno teško iščitati. Zbog toga su prikazane grafički na slici 5.8. u nastavku.



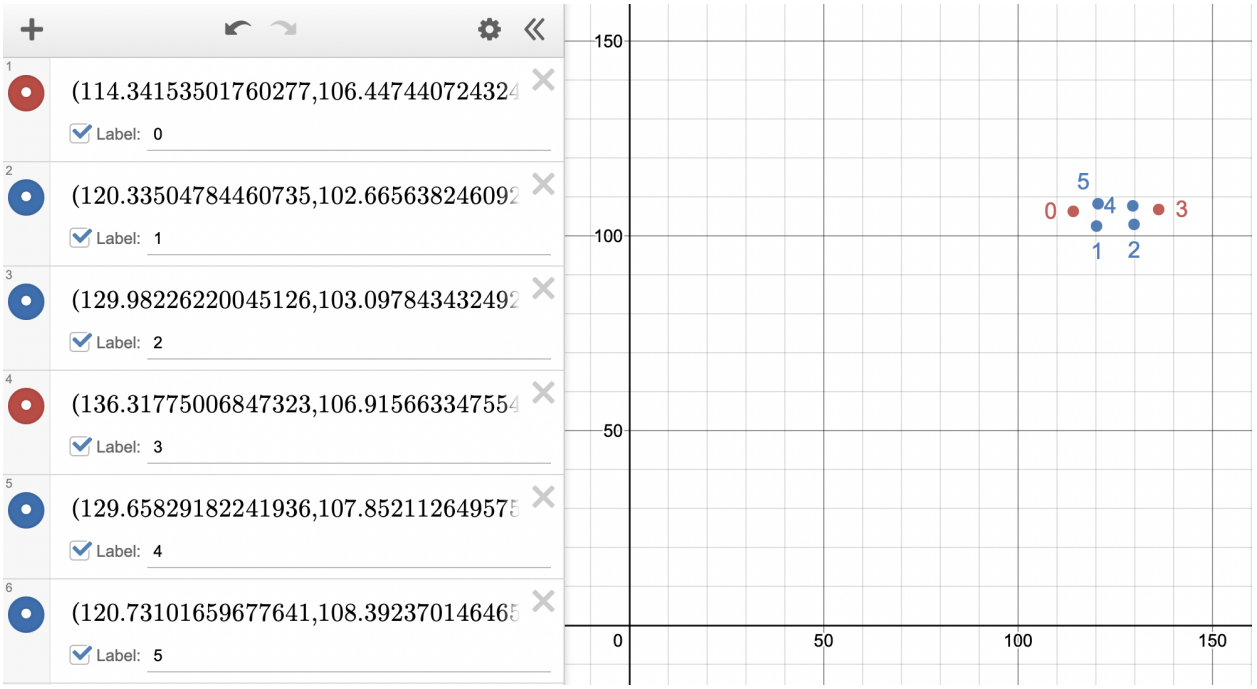
Slika 5.8. Grafički prikaz značajki

Budući da su na slici prikazane sve značajke, bilo bi nezgrapno objašnjavati sve odjednom, pa je stoga u nastavku objašnjena procedura za lijevo oko, dok je za ostale ista ili slična. Ono što se vidi u aplikaciji prikazano je na slici 5.9.



Slika 5.9. Detekcija i iscrtavanje značajki oka

S gornje slike to nije vidljivo, ali okvir oko oka je iscrtan pomoću 6 točaka. Koordinate tih točaka su prikazane na slici 5.10. u lijevom dijelu, dok desni dio predstavlja grafički prikaz tih koordinata. Navedeno je odrađeno za svaku skupinu značajki, te se zaključilo koje točke su ključne. Za lijevo oko je vidljivo kako su ključne točke 0 i 3, te su one označene crvenom bojom.



Slika 5.10. Grafički prikaz značajki lijevog oka

Navedene značajke su ključne jer se iz njih izvlači širina oka. Navedena širina se dijeli sa širinom okvira, te se taj omjer sprema.

$$leftEyeWidthRatio = \frac{leftEye.Points[3].x - leftEye.Points[0].x}{FaceBoundingBox.width} \quad (5\text{-}1)$$

Za oko je tu riječ o širini, dok je za neke druge značajke riječ o visini. Pri dnu poglavlja je popis svih značajki koje se prate i spremaju. Računa se omjer širina, odnosno visina te se kao takav sprema, jer će biti isti na svim uređajima. Kada su svi omjeri izračunati, kreira se objekt po klasi *FaceFeatures*, te se pridružuje uz korisnika ako je riječ o registraciji ili uspoređuje s dohvaćenim ako je riječ o prijavi. Navedeni omjeri se spremaju prilikom registracije korisnika, dok se prilikom prijave dohvaćaju i uspoređuju s trenutnim. Dozvoljeno je određeno odstupanje budući da se omjeri spremaju s dosta decimala i nije moguće da budu potpuno isti. Formulom (5-2) je prikazan izračun odstupanja, a empirijski je određeno da trenutna značajka smije odstupati 3% od spremljene.

$$odstupanje = \frac{|currentFeature - savedFeature|}{savedFeature} \quad (5\text{-}2)$$

Za sve omjere korištena je širina odnosno visina vanjskog okvira, te potrebna dimenzija od značajke. Značajke koje su korištene su: širina lijevog i desnog oka, širina i visina nosa, širina i visina usana, širina i visina kontura lica, te razmak između očiju i obrva. Kao par značajki kojima je moguće pristupiti se navodi i zjenice, međutim unutar dokumentacije se navodi kako to programski okvir još uvijek ne prepoznaje, nego se ona postavlja u središte oka, te kao takva nije od koristi. Moguće je da u budućim verzijama biblioteka i iOS-a broj značajki bude povećan, te stoga bude omogućena još koja ključna informacija.

5.2.5. Testiranje

Testiranje je obavljeno ručno. Najviše je testiran sam rad prepoznavanja lica, jer je to glavna tema diplomskog rada. Prilikom testiranja korišteno je više osoba, a rezultati su prikazani na tablici 1. Svaka od navedenih osoba se pokušala prijaviti kao neka druga 25 puta.

	Nikola	Iva	Matej	Renato
Nikola	0.89	0.03	0.01	0.05

Iva	0.02	0.87	0.02	0.04
Matej	0.03	0.02	0.85	0.01
Renato	0.1	0.04	0.02	0.85

Tablica 1. Usporedba rezultata

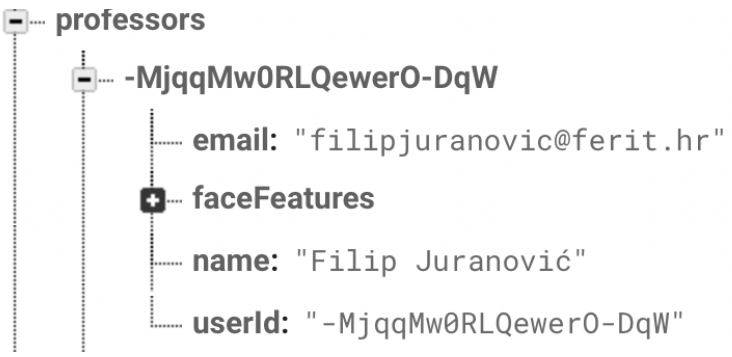
Na tablici 1 koja je prikazana vidi se usporedba rezultata za testirane 4 osobe. Budući da je aplikacija koncipirana tako da se odabere koja se osoba želi prijaviti, lijevo je prikazana odabrana osoba, dok je gore prikazana osoba koja se pokušava prijaviti. Kao što se vidi na dijagonali matrice ako su te dvije osobe iste uspješnost je zadovoljavajuća. Međutim ako aplikacija nije uspjela, ne znači i da će pogriješiti. ako ne prepoznaje osobu i dalje će pokušavati, a neće prepoznati nekoga drugog. Međutim vidljivo je da ponekad je moguće da se druga osoba prijavi, no taj postotak je ispod 5%, a što je moguće i zbog određenog dozvoljenog odstupanja. Budući da među testiranim osobama ima i sličnosti u praksi bi taj broj bio još manji. Zapravo je ova tablica na svojstven način pokazatelj i koliko dvije osobe sliče jedna na drugu. Vidljivo je i da za korisnika Matej je najmanja šansa pogreške, a moguće je da je navedeno zbog toga što nosi naočale.

5.3. Baza podataka u stvarnom vremenu

U ovom poglavlju opisan je model baze podataka u stvarnom vremenu. Za izradu se koristio Firebase.

5.3.1. Spremanje podataka

Prilikom registracije bilo kojeg tipa korisnika podatci o njemu se spremaju u bazu podataka. U ovisnosti o korisniku se spremaju različiti podatci poput imena, prezimena, elektronične pošte i ostalog, dok se za svakog korisnika sprema jednak broj značajki lica. Na slici 5.11. je prikazan jedan profesor unutar baze podataka. Spremanje podataka se za svakog korisnika događa samo jedan put, dok je za profesora i studenta moguće ponovno uzimanje i spremanje podataka ako kontaktiraju administratora. Osim podataka za korisnike, spremaju se i nazočnosti za studente, kao i kolegiji za svakog profesora. Spremanja nazočnosti za studente se događaju tek na kraju kada profesor potvrdi kraj prikupljanja, a ne u trenutku kada se student i prepozna. Prikaz spremanja podataka za profesora dan je na slici 5.12.



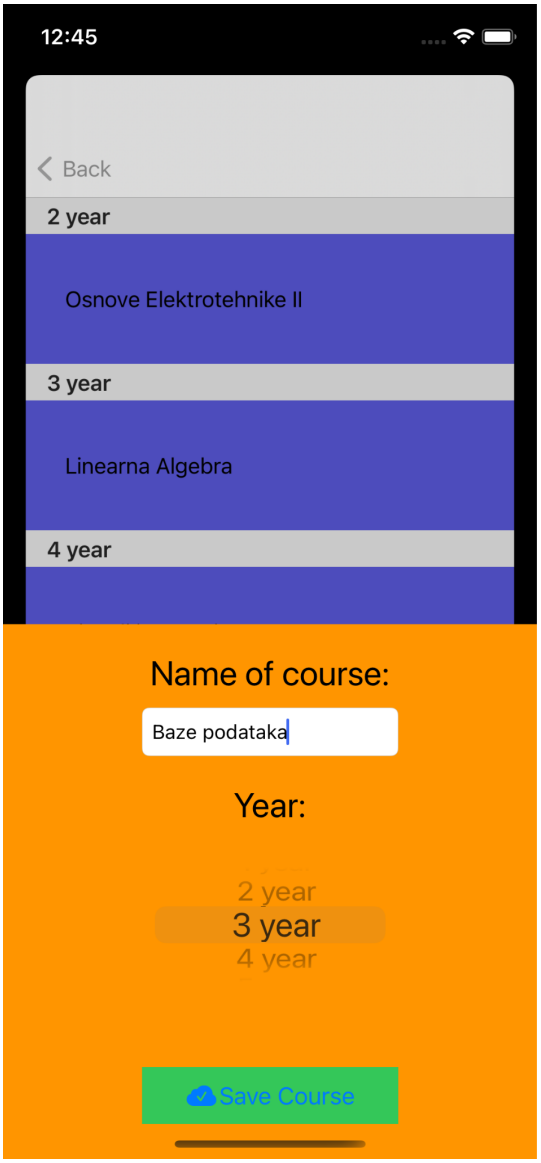
Slika 5.11. Prikaz novog profesora unutar baze podataka

```
extension AddNewCourseViewController {

    @IBAction func saveCourseButtonActionHandler(){
        guard let professor = professor else { return }

        let randomId: String? = Database.database().reference().childByAutoId().key
        guard let randomId = randomId else { return }

        let keyYear: String = years[courseYearPickerView.selectedRow(inComponent: 0)]
        let courseName: String = courseNameTextField.text!
        let course = Course(professorId: professor.userId, courseName: courseName, courseId: randomId)

        ref.child("courses").child(keyYear).setValue([randomId : course.getAsDictionary()]){
            (error:Error?, ref:DatabaseReference) in
            if let error = error {
                print("Data could not be saved: \(error).")
            } else {
                print("Data saved successfully!")
            }
        }
        dismiss(animated: true, completion: nil)
    }

    @IBAction func courseNameDidChangeActionHandler(){
        if let lenght = courseNameTextField.text,
           lenght.count > 3 {
            saveCourseButton.isEnabled = true
            saveCourseButton.alpha = 1
        }
        else{
            saveCourseButton.isEnabled = false
            saveCourseButton.alpha = 0.5
        }
    }
}
```

Slika 5.12. Spremanje novog kolegija u bazu podataka

Na slici 5.12. je prikazano spremanje novog kolegija u bazu podataka, a kako izgleda u aplikaciji je vidljivo na slici 5.13. Tipku za spremanje nije moguće stisnuti dok se ne unese ime kolegija i odabere godina. Za svaki kolegij se u bazi osigurava novi nasumični identifikacijski broj. Unutar čvora courses pod čvorom odabrane godine se kreira novi čvor s unikatnim identifikatorom, gdje

biva spremljen novi kolegij. Po završetku se ispisuje poruka je li uspješno obavljeno, te se modalni prozor zatvara.



Slika 5.13. Spremanje novog kolegija u bazu podataka

5.3.2. Dohvaćanje podataka iz baze

U ovisnosti od korisnika i točke gdje se korisnik nalazi unutar aplikacije odvija se dohvaćanje podataka iz baze podataka. Za profesora i asistenta podatci se uvijek dohvaćaju pojedinačno, kao i za studenta ako samostalno vrši prijavu. Međutim ako profesor otvori novo predavanje dohvaća se lista podataka za sve studente kako bi se verifikacija studenata obavila što brže. Osim navedenog iz baze podataka se dohvaćaju popisi studenata, profesora, asistenata, kao i popisi

predavanja, ali i nazočnosti studenata. Na slici 5.14. je prikazano dohvaćanje i prikaz podataka za profesore.

```
override func viewDidLoad() {
    super.viewDidLoad()

    guard let professor = professor else { return }

    ref = Database.database().reference()
    databaseHandle = ref.child("courses").observe(.childAdded, with: { snapshot in

        snapshot.children.allObjects.forEach { item in

            if let child = (item as? DataSnapshot){

                let course = Course(withSnap: child)

                if course.professorId == professor.userId {
                    if !self.sections.contains(snapshot.key){
                        self.sections.append(snapshot.key)
                    }
                    self.courses[snapshot.key, default: []].append(course)
                }
            }
        }
        self.professorCoursesTableView.reloadData()
    })
    professorCoursesTableView.delegate = self
    professorCoursesTableView.dataSource = self
}
```

Slika 5.14. Prikaz kolegija profesora

Svi kolegiji se nalaze na bazi podataka pod čvorom "*courses*". Taj čvor se osluškuje, te se podatci ažuriraju ako dođe do neke promjene. Unutar tog čvora, kolegiji su razvrstani po godinama, pa tako svaki snapshot predstavlja po jednu godinu. Napisani kod će se izvršiti 5 puta, jer u bazi postoji 5 čvorova koji predstavljaju godine. Za svaku godinu će se, za svaki od njenih objekata pokušati s tim objektom napraviti novi objekt tipa Kolegij. Nakon toga se provjerava pripada li navedeni kolegij tom profesoru, odnosno je li profesorov identifikacijski broj isti kao onaj koji je zapisan unutar kolegija. ako jest, kolegij se dodaje u polje kolegija i prikaz se osvježava.

5.4. TRENUTNA I MOGUĆA POBOLJŠANJA

U ovom dijelu diplomskog rada su opisana trenutna poboljšanja koja su izvediva u ovoj fazi aplikacije, ali je potrebna dodatna akcija od profesora. Također, opisana su i moguća poboljšanja, koja nisu dostupna u ovoj verziji aplikacije ali bi se mogla uvesti u nekoj od narednih verzija u svrhu poboljšanja korisničkog iskustva, sigurnosti ili točnosti.

5.4.1. Poboljšanje sigurnosti

U svrhu poboljšanja sigurnosti rada aplikacije daju se dvije preporuke kojih bi se profesor i asistent trebali pridržavati.

5.4.1.1. Vođeni pristup

U trenutku kada profesor pokrene prijavu za novo predavanje izbacit će mu se sigurnosno upozorenje u vidu skočnog prozora da bi bilo sigurno i pametno da na svom uređaju omogući vođeni pristup. Riječ je o mogućnosti ograničavanja uređaja na samo jednu aplikaciju. Moguće je ograničiti fizičke gumbe koje korisnik može koristiti, kao i geste, a čak i dijelove zaslona koje može koristiti.[10] Pametno je osigurati da neki od studenata slučajno ili namjerno ne zatvori aplikaciju ili da ne pristupa privatnim podatcima na uređaju od profesora. S navedenim se jako povećava sigurnost, ali se u prosjeku povećava i brzina, jer se ne može dogoditi da bilo tko zatvori aplikaciju i izgubi podatke prikupljene do tada. Profesoru ovo ne oduzima vrijeme i nakon napomene se očekuje da poštuje obavijest. S trenutnim pravima ovo nije moguće izvesti programski. Nakon Appleovog odobrenja, uređaj se mora staviti u nadzirani način rada.[11] To se postavlja prilikom početnog postavljanja uređaja i označava da uređaj pripada organizaciji. Trenutno ovo nije izvedivo, ali ako bi fakultet osigurao opremu, ovo poboljšanje bi se u narednoj verziji integriralo u aplikaciju.

5.4.1.2. Lozinka za administratora

Ako aplikacija dođe do faze gdje više ne može prepoznati određenog profesora ili studenta, to i nije toliki problem budući da administrator može ponovno snimiti njihove značajke. Međutim ako se dogodi da aplikacija više ne prepoznaje administratora, te ako je on jedini administrator njemu nema tko ponovno postaviti značajke. Zbog ovoga se u svrhu poboljšanja sigurnosti administratoru javlja posebno generirana lozinka koju je potrebno da čuva

radi pristupa u slučaju ako ga aplikacija ne prepozna. Trenutno navedeno nije omogućeno za profesore i studente iz već objašnjenih razloga.

5.4.1.3. Prebrojavanje studenata

Nakon što zadnji student završi s potvrdom svog identiteta, te vrati profesoru mobitel, profesor prije nego što potvrdi i zaključa podatke o nazočnosti, vidi broj studenata koji su uspješno prepoznati, te ako je ranije prebrojao studente može primijetiti pokušaj prevare te poništiti sve. Ova mogućnost se ostavlja profesoru na volju, ali zahtjeva određenu dodatnu akciju.

5.4.2. Poboljšanje točnosti

U svrhu poboljšanja točnosti postoje određeni dijelovi programskog okvira Vision kojima se ne može pristupiti bez slanja zahtjeva Appleu. To u ovoj fazi nije napravljeno te se ostavlja mogućnost za uvođenje u nekoj od sljedećih verzija. Također, poboljšanje ovog programskog okvira i nove mogućnosti su najavljene za iOS 15 verziju, koja u trenutku razvoja aplikacije još nisu bila dostupna. Trenutno većina iPhone uređaja još uvijek nije opremljena LIDAR senzorima koji bi omogućili prepoznavanje značajki u 3D prostoru, te bi razvoj u tom smjeru uveliko ograničio broj uređaja na kojima se aplikacija može pokrenuti. Zbog toga se ovo poboljšanje ostavlja za budućnost kada se broj uređaja sa senzorom poveća. Aplikacija trenutno ne može biti sigurna da se stvarna osoba nalazi pred njom, jer radi na dvodimenzionalnim slikama. Rješenje ovome problemu koje bi se moglo implementirati već u sljedećoj verziji bi bilo zahtijevanje od korisnika da napravi neku radnju poput otvaranja usta, zatvaranja očiju ili slično. Međutim na veliki broj studenata s ovim korakom bi se izgubilo na brzini.

5.4.3. Poboljšanje brzine

U svrhu poboljšanja brzine u nekoj od narednih verzija bi se omogućilo prijavljivanje studenata s vlastitim uređajem u trenutku kada profesor odobri prijave, za razliku od trenutnog stanja gdje profesor šalje svoj uređaj po učionici. Jedan od glavnih nedostataka ovog poboljšanja je nedovoljan postotak uređaja s iOS sustavom među studentima. Osim navedenog bilo bi potrebno osigurati da se student stvarno nalazi unutar učionice. Navedeno bi se moglo riješiti na više načina, a neki od primjera su upotreba lokacije uređaja, provjera povezane bežične mreže ili povezivanje s profesorovim mobitelom.

6. ZAKLJUČAK

Kroz diplomski rad je obrađena tema izrade iOS mobilne aplikacije za evidenciju dolaznosti studenata na nastavu upotrebom značajki lica za verifikaciju identiteta. Mobilna aplikacija kao takva rješava problem sigurnog vođenja evidencije studenata na nastavi, a omogućuje studentu uvid u svoju dolaznost u svakom trenutku. Izabrana je iOS platforma za izgradnju aplikacije, zbog boljeg programskog okvira za prepoznavanje značajki lica. Koriste se službeni alati i razvojna okruženja od Applea, Firebase baza podataka za spremanje korisnika i njihovih podataka. Većina sličnih rješenja koja postoje su usmjerenija na zaposlenike i evidenciju dolaska i odlaska s posla. Korišten je relativno novi programski jezik Swift, obrazac Model View Controller i programski okvir Vision. Korisničko sučelje je rađeno unutar Storyboarda. Kao glavni element za prikaz popisa je korišten UITableView. Prepoznavanje lica i njegovih značajki se obavlja 30 puta u sekundi, a za svaku sličicu se provjerava postoji li lice na njoj. Ako postoji provjeravaju se značajke lica. Značajke na koje se fokusira su značajke očiju, nosa, usni i konture lica. Svaka od njih ima ključne točke, čiji se podatci obradom spremaju, a kasnije provjeravaju i uspoređuju. Pokretanje i testiranje se radi na fizičkom uređaju. Aplikacija zadovoljava uvjete brzine, sigurnosti i točnosti. Postoji prostor za poboljšanja, ali su za određena poboljšanja potrebna prava koja mora Apple dozvoliti.

LITERATURA

[1] Web stranica AppStore (https://apps.apple.com/us/app/facio-me/id1567336660), posjećeno 21.kolovoza.2021.

[2] Web stranica AppStore (https://apps.apple.com/my/app/jibble-2/id1541142980), posjećeno 19.kolovoza.2021.

[3] Web stranica AloraApp (https://aloraapp.com), posjećeno 18.kolovoza.2021.

[4] Web stranica Apple (https://www.apple.com/ios/ios-14/), posjećeno 17.kolovoza.2021.

[5] Web stranica Apple Developer (https://developer.apple.com), posjećeno 22.kolovoza.2021.

[6] Web stranica Medium (https://medium.com/academy-eldoradocps/vision-framework-for-ios-an-introduction-78d02c3e1ef), posjećeno 25.kolovoza.2021.

[7] Web stranica Newscientist (https://www.newscientist.com/article/2270822-the-number-of-twins-in-the-world-is-the-highest-it-has-ever-been/), posjećeno 23.kolovoza.2021.

[8] Web stranica Apple Developer (https://developer.apple.com/documentation/uikit/uitableview), posjećeno 29.kolovoza.2021.

[9] Web stranica Apple Developer (https://developer.apple.com/documentation/uikit/uinavigationcontroller), posjećeno 29.kolovoza.2021.

[10] Web stranica Apple Support (https://support.apple.com/en-us/HT202612), posjećeno 24.kolovoza.2021.

[11] Web stranica Apple Support (https://support.apple.com/guide/deployment-reference-ios/enabling-device-supervision-ior7ba06c270/web), posjećeno 25.kolovoza.2021.

[12] Web stranica AppStore (https://apps.apple.com/my/app/jibble-2/id1541142980), posjećeno 23.kolovoza.2021.

[13] M. Turk, A. Pentland, Eigenfaces for Recognition, Journal of Cognitive Neurosicence, Vol. 3, No. 1, 1991, pp. 71-86

[14] Irgens, Peter, et al. "An efficient and cost effective FPGA based implementation of the Viola-Jones face detection algorithm." HardwareX 1 (2017): 68-75.

[15] Web stranica Apple Machine Learning (https://machinelearning.apple.com/research/face-detection), posjećeno 03.09.2022

SAŽETAK

Diplomski rad prikazuje proces izrade mobilne aplikacije za bilježenje nazočnosti studenata. Ideja je spriječiti krivotvorenje potpisa, izbaciti upotrebu papira i ubrzati cijeli proces. iOS platforma je odabrana jer ima kvalitetnije biblioteke nego Android. Glavna značajka je prepoznavanje lica, koja se koristi kao alat za identifikaciju i verifikaciju. Kao baza podataka je korišten Firebase, Swift kao programski jezik i Vision biblioteka za detekciju lica i značajki lica. Prikazana su najpopularnija slična rješenja, kao i najpopularniji algoritmi koji se koriste za detekciju lica i nj značajki. Opisane su značajke koje se smatraju ključnim, kako se uzimaju, što se radi s njima, kako se spremaju i kako se provjeravaju. Testiranje je obavljeno ručno i rezultati su prikazani s mogućnostima za napredak.

Ključne riječi: detekcija lica, promatranje lica, značajke lica, iOS aplikacija, nazočnost studenata

ABSTRACT

This thesis demonstrates the process of making a mobile application for recording student attendance. The idea is to prevent signature forgeries, eliminate the use of paper, and speed up the whole process. iOS platform was chosen because it has better quality frameworks than Android. The main feature is facial recognition, which has been used as an identification and verification tool. Firebase was used as a database, Swift as a programming language, and Vision framework for face and facial landmark detection. The most popular similar solutions are shown, as well as the most popular algorithms used for face detection and its features. It is shown which features are considered essential, how they are taken, what is done with them, how they are saved, and how they are checked. Testing was done manually and results are shown with possibilities of improvement.

Key words: face detection, face observation, facial landmark. iOS application, student attendance

ŽIVOTOPIS

Nikola Barbarić rođen je 13.11.1997. godine u Novoj Bili, Bosna i Hercegovina. Pohađao je osnovnu školu u Žepču, nakon koje upisuje srednju školu "tehničar za mehatroniku" u katoličkom školskom centru "Don Bosco Žepče". Nakon srednje škole odlazi u Osijek, gdje 2016. upisuje preddiplomski studij, smjer računarstvo na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija. Nakon preddiplomskog studija, na istom fakultetu upisuje i diplomski studij Računarstva, smjer Informacijske i podatkovne znanosti. Nakon android aplikacije, koju je radio za završni rad, na diplomskom studiju otkriva iOS kao novu strast, te nastavlja u tom smjeru.

_____

Potpis autora