Chat aplikacija za iOS platformu

Šaravanja, Kristian

Undergraduate thesis / Završni rad

2019

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:905824

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2025-03-24**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERAU OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Stručni studij informatike

iOS ChatAplikacija

Završni rad

Kristian Šaravanja

Osijek, 2018.

SADRŽAJ

1. UVOD 1
1.1 Zadatak završnog rada 1
2. PROGRAMSKA ARHITEKTURA I KORIŠTENI PROGRAMSKI ALATI 2
2.1 Korištene tehnologije i alati 2
2.1.1 Swift programski jezik 2
2.1.2 Platforma Firebase 3
2.1.3 Xcode razvojno okruženje 4
2.2Arhitektura aplikacije 6
2.2.1 MVVM arhitektura 6
2.2.2 Kordinator 7
3. PROGRAMSKO RJEŠENJE 10
3.1Registriranje korisnika u bazu podataka 10
3.2 Razgovor u stvarnom vremenu (Real-time) 12
3.3 Razvoj korisničkog sučelja 14
4. KORIŠTENJE I IZGLED APLIKACIJE 17
5. ZAKLJUČAK 22
LITERATURA 23
SAŽETAK 24
TITLE 24
ABSTRACT 24
ŽIVOTOPIS 25

1. UVOD

U ovom radu prikazuje se proces izrade mobilne aplikacije koja omogućuje razgovor između dva korisnika. Namjena aplikacije je, omogućiti dopisivanje u stvarnom vremenu (engl. *real time*) između dva međusobno povezana korisnika. Svaki korisnik aplikacije može se registrirati i stvoriti svoj profil,slati nove poruke odabranim prijateljima, pretraživati svoje prijatelje, uređivati profil i slično. Rješenje je ostvareno pomoćualata kao što subaza podataka „Firebase“ i iOS SDK (engl. *software development kit*) implementirani u aplikaciju za mobilni uređaj iPhone. Aplikacija je napisana u programskom jeziku Swift, te je razvijana u iOS okruženju. U prvom poglavlju navedeni su alati i tehnologije koji su omogućili razvitak aplikacije kao što su programska razvojna okruženja, programski jezik te baza podataka Firebase. U drugom poglavlju opisano je programsko rješenje te su detaljno prikazani važni dijelovi programskog koda. Objašnjene su metode kojima se došlo do rješenja. U trećem poglavlju prikazana je gotova aplikacija te je opisano korištenje aplikacije iz perspektive korisnika.

1.1 Zadatak završnog rada

U sklopu završnog rada potrebno je napraviti aplikaciju koja će omogućitidopisivanje u stvarnom vremenu (engl. *real time*) između dva registrirana korisnika. Poruke i korsnike je potrebno zapisati u bazu podataka. Swift i Firebase su tehnologije koje su korištene za izradu aplikacije, a za korištenje je potrebno skiniti aplikaciju i napraviti račun.

2. PROGRAMSKA ARHITEKTURA I KORIŠTENI PROGRAMSKI ALATI

U ovom poglavlju opisani su korišteni programski alati kao što su programski jezik, programski okviri i razvojna okruženja, te predložena programska arhitektura MVVM-C

2.1 Korištene tehnologije i alati

2.1.1 Swift programski jezik

Swift je programski jezik koji je razvijen od strane Chrisa Lattnera iz tehnološke tvrtke Apple Inc. i prvi put je predstavljen 2014. godine na Worldwide Developers Conference (WWDC). Swift je razvijen za iOS, macOS, watchOS, tvOS i Linux sustav. Swift je nasljednik programskog jezika Objective- C što znači da međusobno dijele metode i smatra se velikim unaprijeđenjem u odnosu na svog prethodnika. Swift uvodi dva nova tipa podataka, opcionalne tipove (engl. *optionals*) i produžetke (engl. *extensions*), te ima poboljšane performanse, rukovanje datotekama i memorijom. Uz to, uvedena je i nova sintaksa jezika koja nalikuje običnom engleskom jeziku i vrlo je prijateljska za developere početnike. Strogo je tipski jezik što znači da jasno definira tip svake varijable ili objekta. ARC(*Automatic reference counting*) koji služi za praćenje i upravljanje memorijom aplikacije, čisti memoriju ukoliko određeni dio nije potreban i više se ne koristi.

```
// MARK: - Model serialization

extension User {

    init?(dictionary: [String: Any], id: String) {
        guard
            let name = dictionary["name"] as? String,
            let email = dictionary["email"] as? String,
            let profileImage = dictionary["profileImage"] as? String
            else {
                return nil
        }

        self.name = name
        self .email = email
        self .profileImage = profileImage
        self.id = id

    }
```

Slika 2.1.1 Primjer Swift koda

Swift se može koristiti kao objektno-orijentiran ili funkcionalan programski jezik. Još jedna njegova odlika je ta da je i protokolno orijentiran programski jezik (engl. *protocol oriented*). Protokoli, u drugim jezicima nazivani sučeljima (engl. *interface*), definiraju nacrt metoda, svojstava i drugih zahtjeva koji odgovaraju određenom zadatku ili komadu funkcionalnosti. Klase i strukture koje se prilagode protokolu jamče da će implementirati ta svojstva i metode. Verzija Swifta korištena u aplikaciji je Swift 4.

2.1.2 Platforma Firebase

U sklopu završnog rada korišten je poslužitelj Firebase. Firebase je tehnologija koja omogućava izradu mobilnih i web aplikacija bez programiranja na strani poslužitelja. Firebase Realtime Database je NoSQL baza podataka koja omogućuje pohranu i sinkronizaciju između korisnika u stvarnom vremenu.Baza podataka u stvarnom vremenu zapravo je samo jedan veliki JSON objekt kojim programeri mogu upravljati. JSON (JavaScript Object Notation) je jedan od tekstualnih otvorenih standarda dizajniran za čitljivu razmjenu podataka. Firebase poduzima sve potrebne korake: pohranjivanje podataka, potvrđivanje korisnika i primjenjuje pravila pristupa. Podržava web, iOS, OS X i Android klijente.

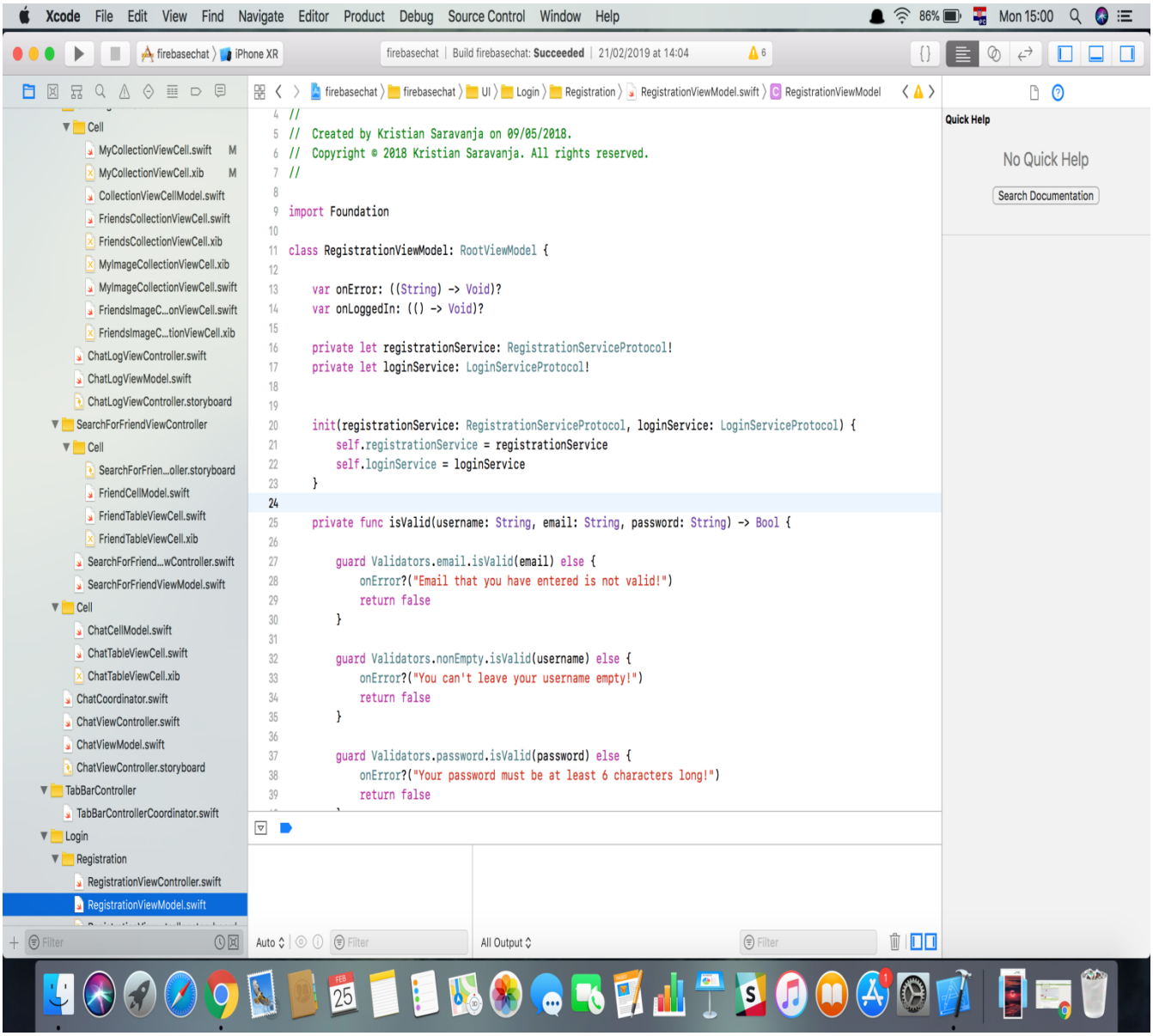
Slika 2.1.2 Izgled baze podataka Firebase

Ključne mogućnosti Firebase-a su:

1. Real time - Umjesto tipičnih HTTP zahtjeva, Firebase koristi sinkronizaciju podataka - svaki put kada se podaci promijene, svaki povezani uređaj prima to ažuriranje u nekoliko milisekundi
2. Offline - Firebase aplikacije ostaju „responzivne“ čak iu izvanmrežnom načinu rada jer Firebase Realtime Database SDK (engl. *software development kit*) održava vaše podatke na disku. Nakon ponovnog uspostavljanja veze, klijentski uređaj prima bilo kakve promjene koje je propustio, sinkronizirajući ga s trenutnim stanjem poslužitelja.
3. Dostupno iz klijentskih uređaja - Firebase bazi podataka može se pristupiti izravno s mobilnog uređaja ili web preglednika; nema potrebe za poslužiteljem aplikacija. Sigurnost i provjera valjanosti podataka dostupni su putem Firebase sigurnosnih pravila baze podataka u realnom vremenu, pravila temeljenih na izrazima koja se izvršavaju kada se podaci čitaju ili pišu.

2.1.3 Xcode razvojno okruženje

Razvojno okruženje korišteno za izradu aplikacije za MacOS operacijski sustav naziva se Xcode. Razvijen je od strane tehnološke tvrtke Apple.Inc. Podržava razvoj programskih rješenja za MacOS, iOS, watchOS i tvOS. Trenutna verzija Xcodea je 10.1. Također podržava razne programske jezike, kao što su Java, Python, Ruby, ResEdit, C, C++ te iOS nativne jezike Swift i Objectiv C. Sadrži graditelja sučelja (engl. *interface builder*), uređivača izvornog koda (engl. *source editor*) te mnoge druge elemente koji se koriste prilikom razvoja aplikacije, kao što su ugrađeni alati za simulaciju iOS aplikacije na Iphone i Ipad uređajima.



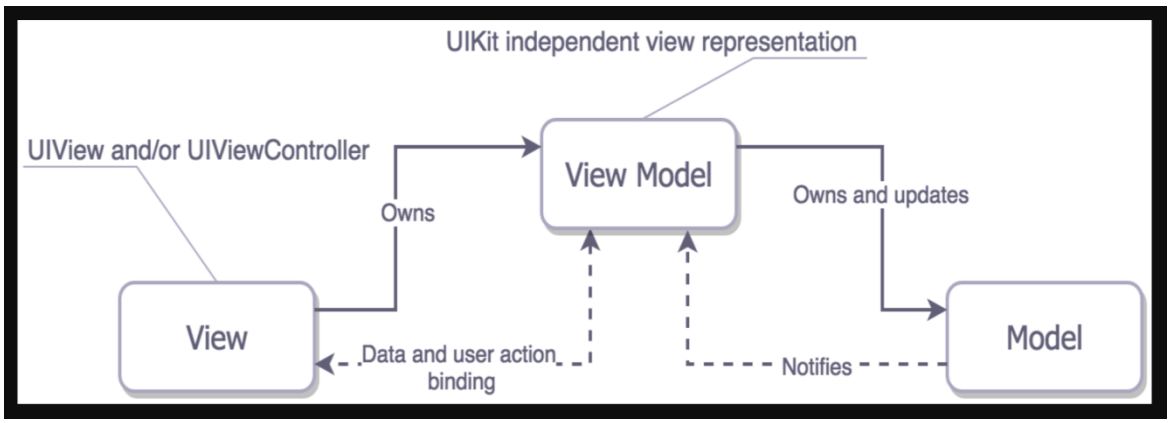
Slika 2.1.3 Primjer razvojnog okruženja Xcode

2.2 Arhitektura aplikacije

Za arhitekturu aplikacije korištena je MVVM-C ili Model-View-ViewModel-Coordinator programska arhitektura. Ona se sastoji od Model-View-ViewModel arhitekture i objekta koordinatora koji služi za navigaciju između kontrolera pogleda, odnosno različitih zaslona aplikacije.

2.2.1 MVVM arhitektura

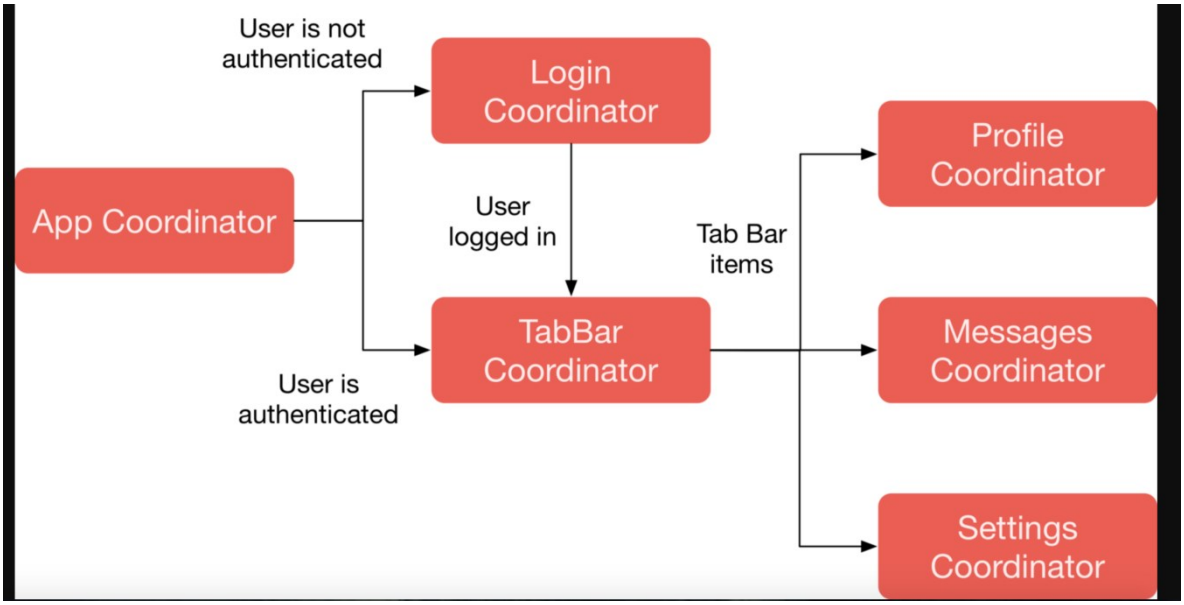
Model-View-ViewModel ili MVVM je programska arhitektura koja sastojise od modela, pogleda (engl. *view*) i pogleda modela (engl. *view model*). U iOS okruženju pogled predstavlja kontroler pogleda (engl. *view controller*) u koji se piše sav kod koji je vezan uz korisničko sučelje, to jest vanjski izgled aplikacije. Model predstavlja modele objekata iz baze, a model pogleda sadrži svu logiku koja povezuje promjene na modelu sa događajima na kontroleru pogleda te se adekvatno tome događaji ažuriraju. Pogled i model nemaju čvrstu vezu između sebe te model pogleda ne zna ništa o pogledu što nam omogućuje lakše testiranje logike koja se nalazi u modelu pogleda. U usporedbi sa MVC arhitekturom, MVVM ima jasno definirane dijelove arhitekture koji vrše određene funkcije. Obično MVVM ima malo više programskog koda, ali pomaže u sprječavanju nastanka ogromnih kontrolera pogleda(engl. *massive view controller*).



Slika 2.2.1 Primjer MVVM arhitekture

2.2.2 Kordinator

Ovaj je uzorak prvi put predstavio iOS zajednici Soroush Khanlou na svom blogu i tijekom svoje prezentacije na NSSpain konferenciji. Ideja koordinator uzorka je stvoriti zasebnu cjelinu koordinatora koja je odgovorna za navigaciju u aplikaciji. Koordinator je zapravo objekt koji upravlja navigacijom između različitih zaslona aplikacije. Postoji jedan koordinator više razine koji upravlja navigacijom cijele aplikacije te on stvara druge koordinatore kao svoju djecu. Koordinatori se trebaju rasporediti po logički povezanima cjelinama i svaki je zadužen za jedan dio aplikacije. Ovim postupkom izdvajamo kontrolere pogleda kao zasebne cjeline koje znaju samo za sebe i za niti jedan drugi kontroler pogleda. Upravo zbog tog razloga koordinator kontrolere pogleda čini pogodnim za višekratno korištenje (engl. *reusable*) te se mogu stvoriti i prikazati iz bilo kojeg dijela aplikacije.



Slika 2.2.1 Primjer koordinator uzorka

```
final class MainCoordinator:  Coordinator {

    private let persistanceService = ServiceLocator.factory.persistanceService
    var childCoordinators: [Coordinator] = []

    enum StartingController {
        case notLoggedIn
        case loggedIn
    }

    private var startingController: StartingController = .notLoggedIn
    private var startingViewController: UIViewController!

    func start() -> UIViewController {

        if persistanceService.isUserLoggedIn {
            startingController = .loggedIn
        } else {
            startingController = .notLoggedIn
        }

        let vc: UIViewController = {
            switch startingController  {
            case .notLoggedIn:
                return createAuthentication()
            case .loggedIn:
                return createTabBar()
            }
        }()

        startingViewController = vc
        vc.showAsRoot()
        return vc
    }

    private func createAuthentication() -> UIViewController {
        let authCoordinator = AuthenticationCoordinator()
        authCoordinator.parentCoordinator = self
        childCoordinators.append(authCoordinator)

        authCoordinator.showTabBarCoordinator = { [weak self] in
            self?.createTabBar()
```

Programski kod 2.2.2 Primjer programskog koda klase MainCoordinator

Klasa MainCoordinator prikazana prikazom koda 2.2.3 je glavni objekt navigacije u aplikaciji. MainCoordinator klasa nasljeđuje protokol Coordinator koji na sebi ima samo jednu metodu *start*koja je zaslužna za pokretanje prvog zaslona aplikacije. Metoda ima povratni tip *UIViewController*koji je osnovni dio UIKit okvira. UIKit je programski okvir stvoren od strane Apple-a kako bi programeri mogli upravljati i graditi korisničko sučelje aplikacije. Svaki pogled iOS aplikacije je tipa *UIViewController*ili neka njegova podklasa. Metoda *start* vraća kontroler pogleda, ovisno o tome je li korisnik već registriran i logiran ili

ne. Ukoliko je korisnik logiran metoda *createTabBar*vratit će *UITabBarController* kao kontroler pogleda koji sadrži tri instance *UIViewControllera*, a ako je ovo korisniku prvo pokretanje aplikacije ili nije ostao logiran, metoda *createAuthentication* vratit će *UIViewController* koji prikazuje registracijski zaslon koji jeujedno i početni zaslon aplikacije.

```
protocol Coordinator: class {

    @discardableResult
    func start() -> UIViewController
}
```

Programski kod 2.2.3 Coordinator protokol

Kontroler pogleda, u ovom primjeru RegistrationViewController koji nastaje u koordinator klasi AuthenticationCoordinator, zove metodu *instance* koja poziva metodu *createFromStoryboard*, koja omogućuje stvaranje kontrolera pogleda iz storyboarda. Metoda *instance* ima povratni tip *Self*, što znači da vraća opći (engl. *generic*) tip iz metode *createFromStoryboard*, a ona vraća opći tip *UIViewController* podklase kojoj je ime jednako imenu storyboard datoteke. Metoda instance prikazana je u prikazu programskog koda.

```
class func instance() -> Self {
    if let vc = createFromStoryboard(type: self) {
        return vc
    } else {
        print("WARNING: can't create view controller from storybard:\(self)")
        return self.init()
    }
}

private class func createFromStoryboard<T: UIViewController>(type: T.Type) -> T? {

    let storyboardName = String(describing: type)

    let bundle = Bundle(for: T.self)

    guard bundle.path(forResource: storyboardName, ofType: "storyboardc") != nil else {
        return nil
    }

    let storyboard = UIStoryboard(name: storyboardName, bundle: bundle)

    guard let vc = storyboard.instantiateInitialViewController() else {
        print("no vc in storyboard(hint: check initial vc)") ; return nil
    }

    return vc as? T
}
```

Programski kod 2.2.4 *instance* i *createFromStoryboard* metode za kontroler pogle

3. PROGRAMSKO RJEŠENJE

Programsko rješenje ima nekoliko glavnih odlika, a to su spremanje registriranih korisnika sa svim odgovarajućim podatcima kao što su ime, email, slika u bazu podataka Firebase te je omogućen razgovor u stvarnom vremenu koji je također pohranjen u Firebase zajedno sa porukama između dva korisnika.

3.1 Registriranje korisnika u bazu podataka

Prilikom prvog pokretanja aplikacije, korisniku se prikazuje početni zaslon u kojem korisnik da bi pristupio aplikaciji mora unijeti set osobnih podataka kako bi se registrirao. Neregistriranim korisnicima ulaz nije dozvoljen. Nakon što korisnik unese vrijednosti, pritiskom na registracijski gumb (engl. *register button)*korisnik se sprema kao tip korisnik (*user*) u bazu podataka Firebase te kao tip rječnik (engl. *dictionary*) u lokalnu bazu podataka User Defaults. Rječnik pohranjuje poveznice između ključeva i vrijednosti. Svaka vrijednost povezana je jedinstvenim ključem koji služi kao identifikator za tu vrijednost unutar rječnika. Klasa UserDefaults pruža programsko sučelje za interakciju s zadanim sustavom. Sustav zadanih postavki aplikaciji omogućuje prilagođavanje ponašanja tako da odgovarakorisničkim postavkama.

```
//MARK: - UserDefaults

extension UserDefaults: PersistanceServiceProtocol {

    var user: User? {
        get {
            if let userDictionary = object(forKey: Keys.user) as? [String: Any] {
                guard let id = userDictionary["id"] else {
                    return nil
                }
                return User(dictionary: userDictionary, id: "\(id)")
            } else {
                return nil
            }
        }
        set {
            set(newValue?.asDictionary, forKey: Keys.user)
        }
    }
```

Programski kod 3.1.1 Primjer spremanja i dohvaćanja korisnika u lokalnoj bazi User Defaults

Na primjer, možete dopustiti korisnicima da odaberu odrerđeni jezik, promjene temu aplikacije ili u ovom slučajuda korisnik koji je prijavljen u sustav, ostane prijavljen prilikom slijedećeg pokretanja aplikacije te odgovarajuće tome mu je prikazan drugi zaslon aplikacije. Aplikacije pohranjuju te postavke dodjeljivanjem vrijednosti skupu parametara u korisničkoj bazi podataka. Parametri se nazivaju zadanim postavkama jer se uobičajeno koriste za određivanje zadanog stanja aplikacije prilikom pokretanja.

```
func register(username: String, email: String, password: String, profileImage: URL?) {

    guard isValid(username: username, email: email, password: password) else { return }

    let data = RegistrationData(name: username, email: email, password: password, profileImage:
        profileImage)

    onStartedActivity?()
    registrationService.register(data: data, imageURL: profileImage) { [weak self] result in
        switch result {
        case .success:
            self?.onEndedActivity?()
        case .failure:
            self?.onError?("That email has already been taken!")
        }
    }
}
```

Programski kod 3.1.2 Metoda register koja korisnika pohranjuje u bazu podataka Firebase

set(forKey:) metoda na UserDefaults klasi, sprema objekt korisnika tipa User kao rječnik. Pomoću metode asDictionary objekt se pretvara u rječnik kao što je vidljivo izprogramskog koda na slici 3.1.3

```
var asDictionary: [String: Any] {
    return ["name": name,
            "email": email,
            "profileImage": profileImage,
            "id": id]
}
```

Programski kod 3.1.3 Metoda asDictionary koja sprema korisnika kao rječnik

3.2 Razgovor u stvarnom vremenu(Real-time)

Nakon registracije korisnika, korisnik pritiskom na login gumb (engl. *login button*) ulazi u glavno sučelje aplikacije koje se satoji od tri glavna zaslona aplikacije. Prvi zaslon predstavlja praznu tablicu prijatelja, koja će biti popunjena sa odabranim prijateljima koji se mogu izabrati pritiskom na pretrazi gumb (engl. *search button*) koji se nalazi na navigacijskoj traci.Svi registrirani korisnici dohvaćeni su kao prijatelji trenutno logiranog korisnika pomoću metode fetchFriends.

```
func fetchFriends(completion: @escaping (SearchResult<[User]>) -> Void) {
    var users: [User] = []

    DataService.ApiPaths.users.observe(.childAdded, with: { (snapshot) in

        if let dictionary = snapshot.value as? [String: Any] {
            guard let user = User(dictionary: dictionary, id: snapshot.key) else {
                completion(.failure)
                return
            }
            users.append(user)
            completion(.success(users))
        }

    }) { (error) in
        print(error)
        completion(.failure)
    }

}
```

Programski kod 3.2.1 Dohvaćanje svih registrirani korisnika

Prilikom odabira prijatelja za razgovor, prijatelj se šalje objektu razgovora i pomoću metode *create* stvara se objekt razgovora u Firebase bazi podataka. Objekt razgovora sadrži identifikacijske oznake korisnika i prijatelja pod granom (engl. *users*) te granu poruke (engl. *messages*) koja je prvi prvom stvaranju razgovora prazna.

```
func create(conversation: Conversation, completion: @escaping (ConversationResult) -> Void) {

    existingConversations(conversation: conversation) { exists in
        if exists {
            completion(.failure(.isExisting))
            return
        } else if  conversation.participants.last == self.persistanceServiceProtocol.user?.id {
            completion(.failure(.talkingToYourself))
            return
        }
        else {
            let value = ["users": conversation.participants]
            DataService.ApiPaths.chats.childByAutoId().setValue(value) { (error, _) in

                if let err = error {
                    print(err)
                    completion(.failure(.serviceError))
                }
            }
            completion(.success)
        }
        return
    }
}
```

Programski kod 3.2.2 Stvaranje objekta razgovora u bazi podataka Firebase

Unutar metode create nalazi se još jedna metode existingConversations koja provjerava da li pokušavamo stvoriti razgovor koji već postoji ili razgovor sa samim sobom, što nije dozvoljeno. Kako bi metoda existingConversation vraćala pravu vrijednost implementirana je još jedna pomoćna metoda isExisting kojoj je jedina namjena provjeriti da li identifikacijska oznaka prijatelja s kojim želimo stvoriti razgovor već postoji.

```
private func isExisting(newId: String, friendIds: [String]) -> Bool {
    if friendIds.contains(newId) {
        return true
    } else {
        return false
    }
}
```

Slika 3.2.3 Pomoćna metoda isExisting implementirana unutar existingConversations metode

```
private func existingConversations(conversation: Conversation, onComplete: ((Bool) -> Void)?) {
    var friendIds: [String] = []

    DataService.ApiPaths.chats.observeSingleEvent(of: .value, with: { snapshot in

        if snapshot.exists() {
            for snap in snapshot.value as! [String: Any] {
                if let dictionary = snap.value as? [String: Any] {

                    guard let conversationParticipants = dictionary["users"] as? [String] else {
                        return
                    }
                    if conversationParticipants.first == self.persistanceServiceProtocol.user?.id {
                        friendIds.append(conversationParticipants.last!)
                    }
                }
            }

            if self.isExisting(newId: conversation.participants.last!, friendIds: friendIds) {
                onComplete?(true)
                return
            } else {
                onComplete?(false)
                return
            }

        } else {
            onComplete?(false)
        }

    }) { err in
        print(err)
    }
}
```

Programski kod 3.2.4 Metoda existingConversations koja provjerava postoji li već razgovor

3.3 Razvoj korisničkog sučelja

Prilikom pokretanja aplikacije prikazuje se početni zaslon koji se sastoji od početnog kontrolera pogleda iz kojeg možemo ući u glavno sučelje ukoliko stvorimo račun i prijavimo se. Nakon prijave korisnika u sustav prikazuje se glavno korisničko sučelje unutar aplikacije. Sučelje se sastoji od kontrolera pogleda koji je tipa *UITabBarController* koji sadrži druge kontrolere pogleda tipa *UIViewController.* Instanca *tipa UITabBarController* u sebi sadrži varijablu *viewControllers* koja je polje tipa *UIViewController,* kojoj možemo predati stvorene instance tipa *UIViewControllera*. Instanca *UITabBarControllera* sadrži tri instance *UIViewControllera*, a to su ChatViewController, UserProfileViewController iSettingsViewController. To znači da instanca *UITabBarControllera* sadrži tri glavna zaslona.

```
func start() -> UIViewController {
    rootViewController.viewControllers = [
        startChat(),
        startUserProfile(),
        startSettings()
    ]

    let tabBar = rootViewController.tabBar
    tabBar.tintColor = .mediumSeaGreen
    tabBar.backgroundColor = .defaultBarColor
    tabBar.barTintColor = .defaultBarColor
    rootViewController.showAsRoot()
    return rootViewController
}

private func startChat() -> UIViewController {
    let chatCoordinator = ChatCoordinator()
    childCoordinators.append(chatCoordinator)
    let vc = chatCoordinator.start()
    vc.tabBarItem.image = 
    vc.tabBarItem.imageInsets.top = 4
    vc.tabBarItem.imageInsets.bottom = -4
    return vc
}

private func startUserProfile() -> UIViewController {
    let userProfileCoordinator = UserProfileCoordinator()
    childCoordinators.append(userProfileCoordinator)
    let vc = userProfileCoordinator.start()
    vc.tabBarItem.image = 
    vc.tabBarItem.imageInsets.top = 4
    vc.tabBarItem.imageInsets.bottom = -4
    return vc
}

private func startSettings() -> UIViewController {
    let settingsCoordinator = SettingsCoordinator()
    childCoordinators.append(settingsCoordinator)
    let vc = settingsCoordinator.start()
    vc.tabBarItem.image = 
    vc.tabBarItem.imageInsets.top = 4
    vc.tabBarItem.imageInsets.bottom = -4
    return vc
}
```

Stvaranje instanci *UIViewControllera* prikazano je prikazom programskom koda 3.3.1 Kontroleri pogleda prikazani su u donjoj navigacijskoj traci instance

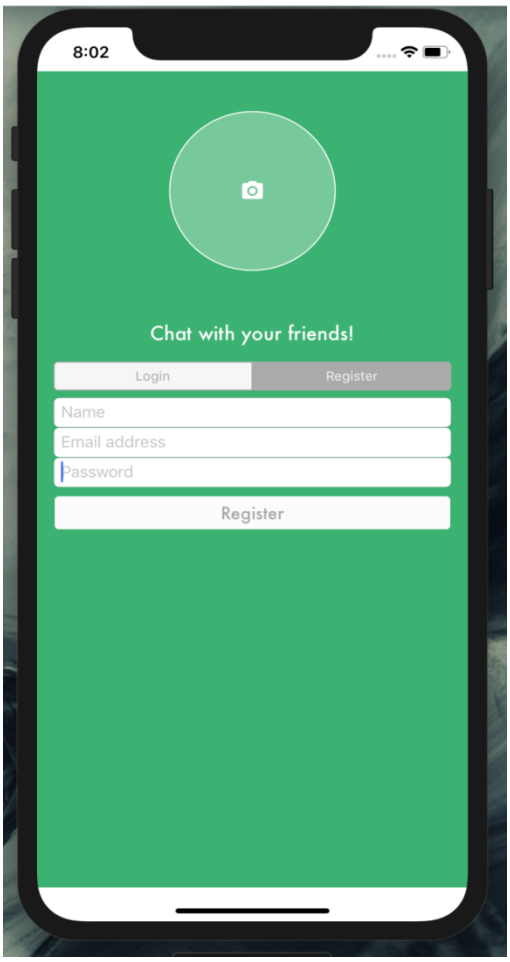
*UITabBarControllera,*navigacija između njih također je osigurana budući da se kontroleri pogleda nalaze u polju *viewControllers* te njihov index odgovara poziciji na donjoj navigacijskoj traci.

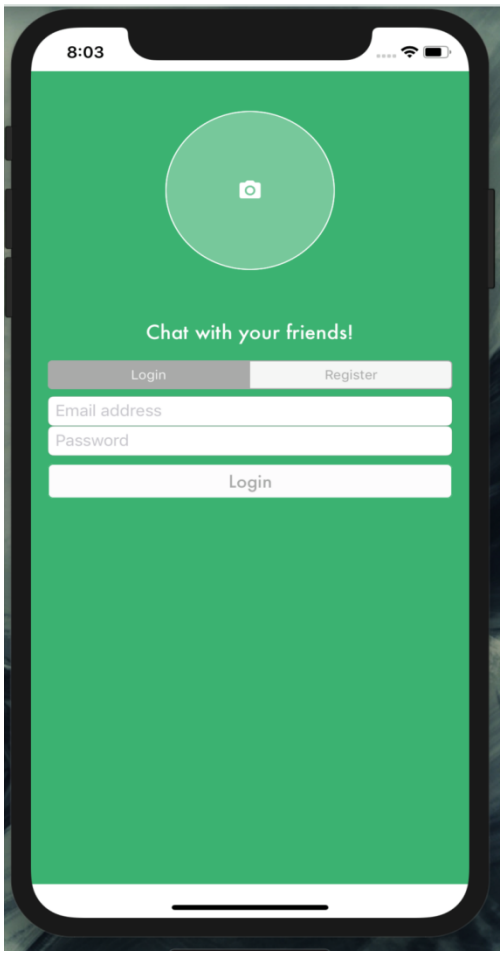
Tablica 3.3.2 Popis sučelja

Sučelje	Početno	Sučelje koje pruža registraciju i prijavu korisniku
Sučelje	Razgovori	Sučelje koje pruža popis trenutnih stvorenih razgovora
Sučelje	Traži prijatelja	Sučelje koje pruža mogućnost pregleda popisa svih registriranih korisnika u sustavu koji koristi aplikaciju
Sučelje	Poruke razgovora	Sučelje koje pruža mogućnost izmjenjivanja tekstualnih i vizualnih poruka između dva korisnika
Sučelje	Biranje pozadine razgovora	Sučelje koje pruža mogućnostbiranja pozadine razgovora
Sučelje	Profil korisnika	Sučelje koje pruža mogućnost upravljanja korisnikovim profilom, kao što su promjena slike i dodavanje opisa te odjava iz sustava
Sučelje	Zaslon postavki	Sučelje koje omogućuje korisniku da vidi neke osnovne podatke unutar aplikacije

4. KORIŠTENJE I IZGLED APLIKACIJE

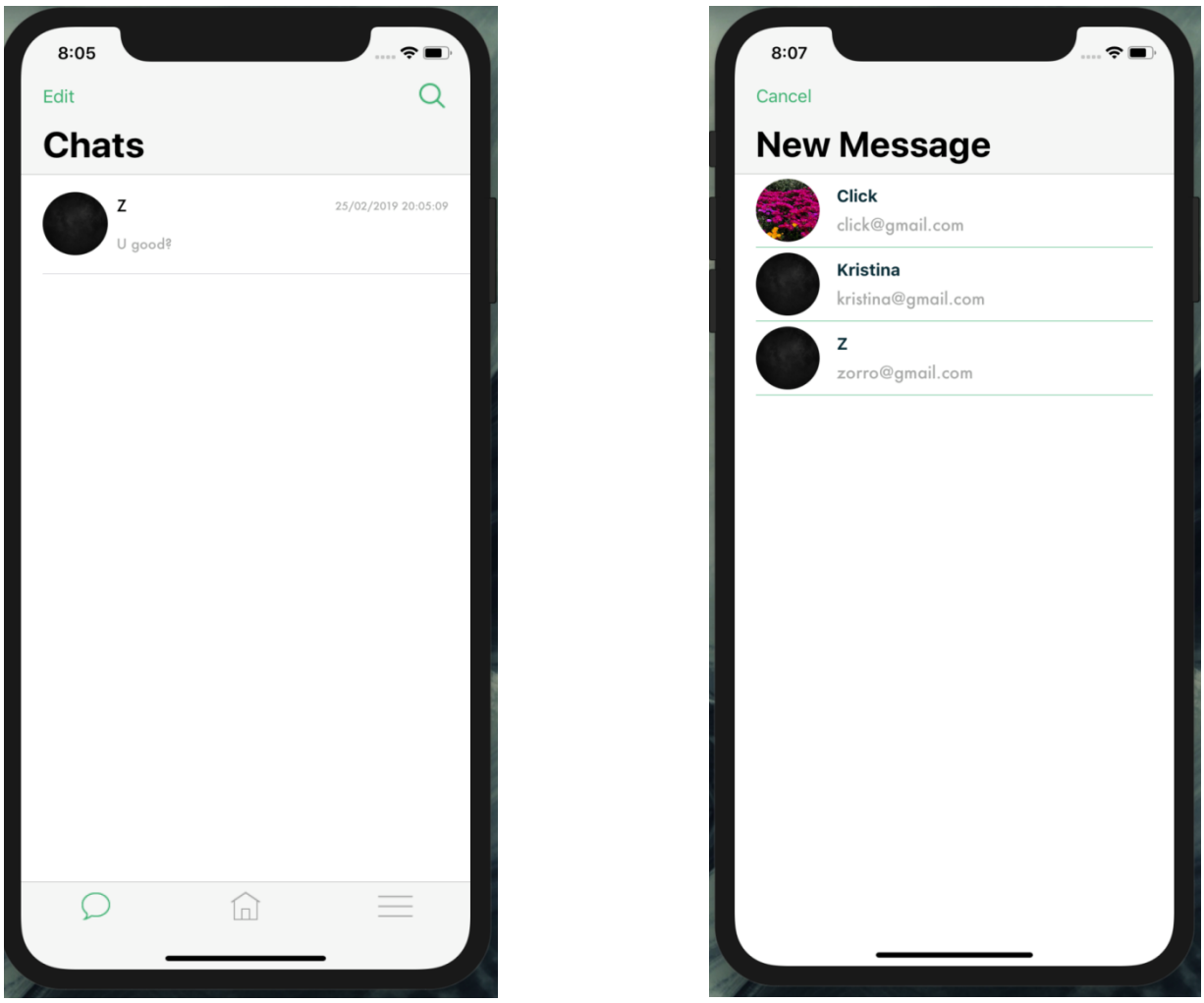
U ovom poglavlju opisano je korištenje i izgled aplikacije. Početni zaslon aplikacije sastoji se od registracijske forme ili od forme za prijavu ukoliko smo se već registrirali u sustav. Promjena formi se vrši pritiskom na gumb Login ili Register. Na početnom zaslonu se još nalaze polja za unos osobnih podataka kao što su korisničko ime, email adresa i lozinka te gumb za odabir slike profila.





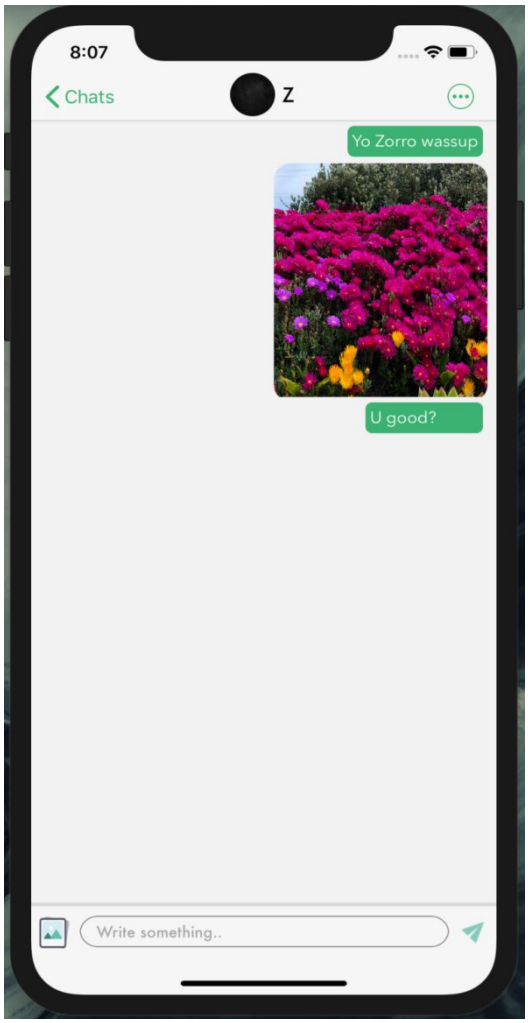
Slika 4.1 Početni zaslon aplikacije

Nakon registracije i prijave u sustav korisniku se prikazuje glavni zaslon aplikacije, koji se sastoji od navigacijske trake (*navigation bar*) prvi vrhu zaslona, koja s lijeve strane sadrži gumb *edit* za uklanjanje postojećih razgovora, te sa desne strane gumb sa slikom povećala koji vodi na zaslon za pretraživanje prijatelja. U sredini se nalazi tablica koja prikazuje razgovore sa prijateljima. Pri dnu zaslona nalazi se traka kartica (*tab bar*) koja služi za navigaciju između većih cijelina aplikacije.

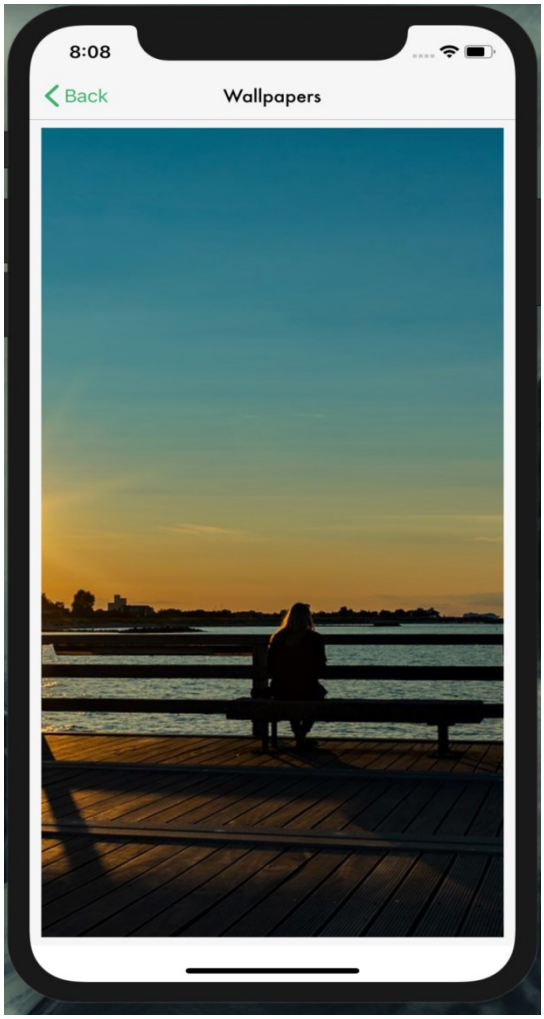


Slika 4.2 Jedan od glavnih zaslona aplikacije koji prikazuje razgovore sa prijateljima (*lijevo*) te zaslon za pretraživanje prijatelja (*desno*)

Pritiskom na razgovor sa nekim prijateljem korisniku se prikazuje zaslon sa dnevnikom razgovora. Na zaslonu u navigacijskoj traci sa desne strane nalazi se gumb sa slikom „*više*“ koji pritiskom vodi korisnika na novi zaslon u kojem ima mogućnost odabira pozadine dnevnika razgovora. S lijeve strane na navigacijskoj traci nalazi se gumb „*nazad*“ koji korisnika vodi na prijašnji zaslon koji je prikazan slikom 4.2. U sredini se nalazi dnevnik razgovora koji prikazuje sve izmjenjene poruke između dva korisnika. Pri dnu zaslona s lijeve strane nalazi se gumb u obliku slike koji služi korisniku za slanje fotografija. Također se nalazi i polje za unos teksta u koju korisnik upisuje željenu poruku koju želi proslijediti svom prijatelju te sa desne strane gumb sa slikom „*pošalji*“ za slanje poruke.

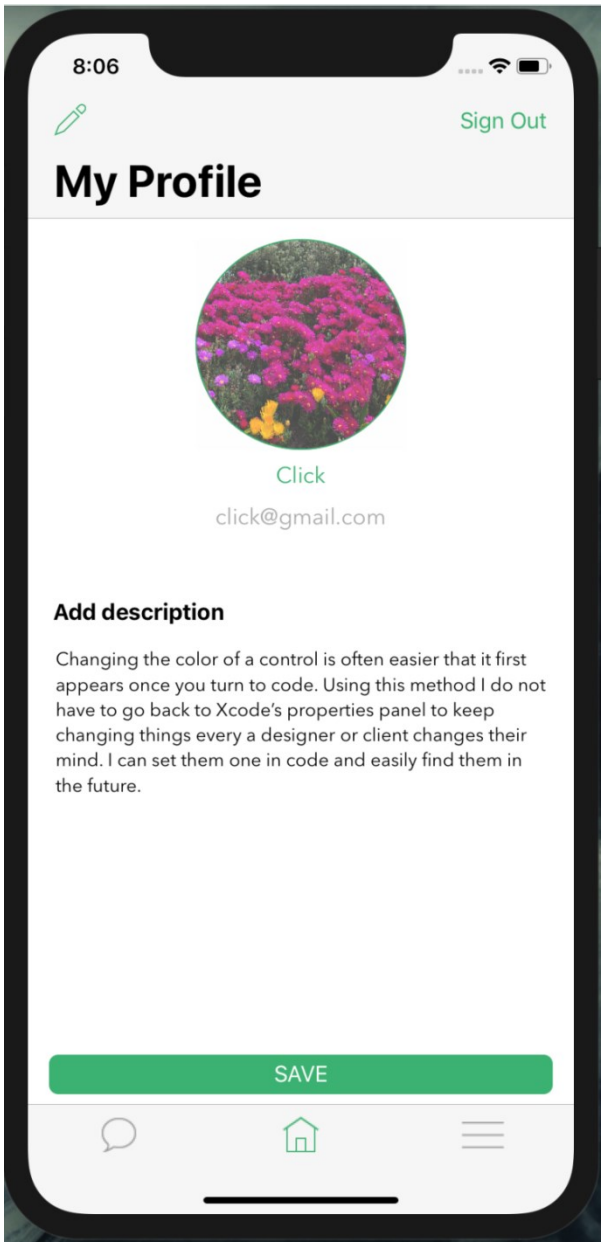


Slika 4.3 Dnevnik razgovora



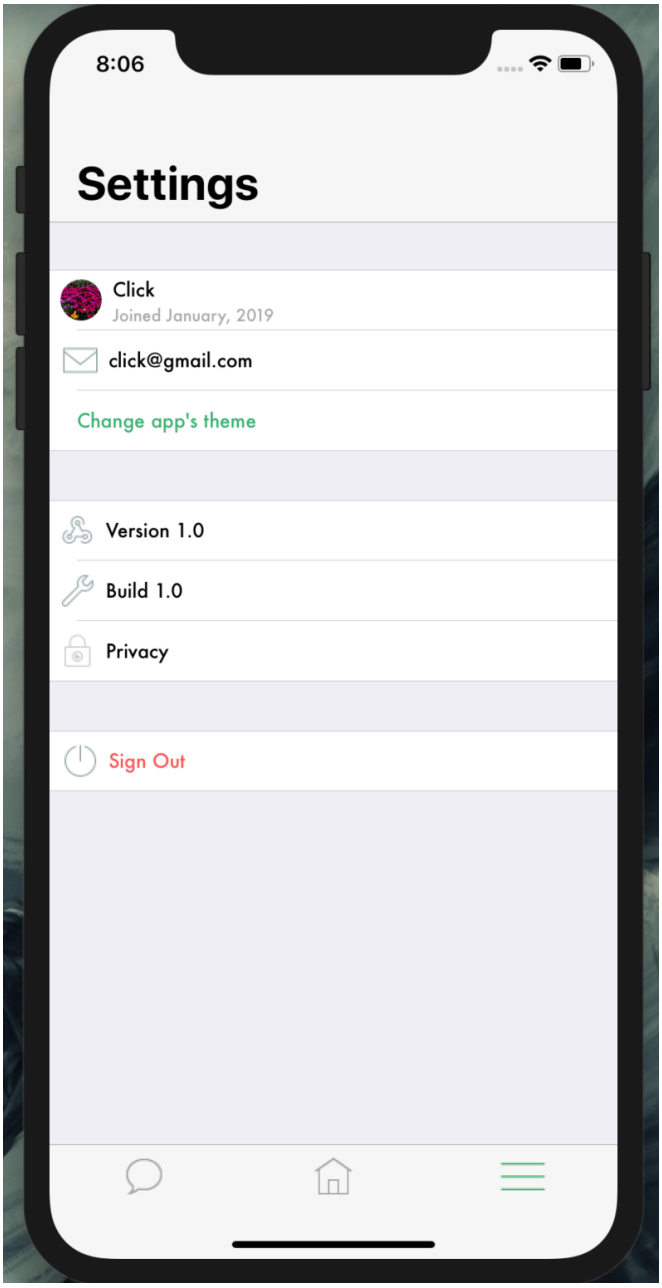
Slika 4.4 Odabir pozadine dnevnika razgovora

Pritiskom na novu karticu na traci kartica (*tab bar*) korisniku se otvara novi zaslon aplikacije koji prikazuje njegov profil. Također u navigacijskoj traci se nalaze dva gumba, s lijeve strane gumb sa slikom „*olovke*“ koji služi za promjenu profilne slike, te gumb sa naslovom „*Odjavise*“ koji služi za odjavu korisnika iz sustava. Prilikom odjave korisniku se prikazuje početni zaslon prikazan na slici 4.1. Na zaslonu profila korisniku su prikazani njegovi osobni podatci koje je unio prilikom prijave kao što su profil slika, korisničko ime te email. Ispod osobnih podataka nalazi se naljepnica „Add description“ koja pritiskom otvara polje za unos željenog opisa te gumb „*SAVE*“ koji služi za spremanje unesenog opisa u bazu podataka.



Slika 4.4 Moj profil

Pritiskom na treću karticu na traci kartica korisniku se otvara novi zaslon koji prikazuje osnovne postavke i informacije o aplikaciji kao što su trenutno logirani korisnik,verzija aplikacije te pravila o privatnosti. Također se prikazuju dva gumba, gumb promjeni temu aplikacije (engl. *change app's theme*) koji korisniku daje mogućnost promjene teme te mogućnost odjave sa gumbom odjava (engl. *sign out*).



Slika 4.5 Postavke

5. ZAKLJUČAK

Ovom aplikacijom želi se omogućiti što učinkovitijei jednostavnije digitalnopovezivanje između dvije osobe u stvarnom vremenu. Aplikacija je napravljena za mobilni uređaj Iphone gdje svaki korisnik može stvoriti račun, pretraživati prijatelje, izmjenjivati poruke s istim, uređivati profil te koristiti još niz ostalih sitnih mogućnosti unutar sustava.

Prilikom izrade programskog rješenja bilo je potrebno dobro postaviti arhitekturu samog sustava te osmisliti arhitekturu baze podataka u koju će biti spremljeni svi podatci. Određivanje i postavljanje pravilne arhitekture svakog sustava ključno je i u samom radu istog.Korištena je Firebase baza podataka u stvarnom vremenu zato što svake promjene u bazi su odmah vidljive i na moblinom uređaju. Aplikacija ima prostora za nadograđivanje dodatnim svojstvima kao što su enkripcija podataka, omogućavanjem slanja video medija te blokiranje prijatelja.

LITERATURA

[1] Swift (*programming language*), https://en.wikipedia.org/wiki/Swift_(programming_language), svibanj 2019

[2] About Swift, https://swift.org/about/, svibanj 2019

[3] The Swift Programming Language(Swift 5.0), Apple Inc., Canada, 2014

[4] Firebase documentation, https://firebase.google.com/docs/ios/setup, svibanj 2019

[5] Introduction to MVVM: Refactoring a MVC app using MVVM design pattern, https://www.appcoda.com/mvvm-vs-mvc/, svibanj 2019

[6] S Khanlou, Coordinators Redux, blog, http://khanlou.com/2015/10/coordinators-redux/, svibanj 2019

[67 Apple Developer Documentation, https://developer.apple.com/documentation/, svibanj 2019

[8] iOS SDK, https://en.wikipedia.org/wiki/IOS_SDK, svibanj 2019

SAŽETAK

Aplikacija koja je napravljena za iOS operativni sustav i mobilni uređaj Iphone razvijana je sa ciljem što bržeg, jednostavnijeg i učinkovitijeg povezivanja dvije osobe u stvarnom vremenu. Za korištenje aplikacije kao i svih njezinih resursa potrebno je samo stvoriti račun te se prijaviti u sustav. Aplikacija je prilagođena korisniku, odnosno jednostavna je za korištenje. Nakon prijave, korisniku se pružaju razne mogućnosti unutar sustava, od kojih je najvažnija komunikacija sa drugim registriranim korisnikom. Kako bi aplikacija bila stabilna i optimalna, implementirano je rukovanje sa greškama.

Ključne riječi: korisnik, račun, komunikacija, iOS aplikacija

TITLE

iOS Chat Application

ABSTRACT

An application that is made for iOS operating system and Iphone mobile device has been developed with the aim of making the real-time communication faster, simpler, and more effective between two people. To use the app as well as all of its resources, you need to create an account and log in to the system. The application is customized to the user, that means, it is easy to use. After logging in, the user is provided with various features within the system, the most important of which is communication with another registered user. In order for the application to be stable and optimal, error handling is implemented.

Keywords: user, account, communication, iOS application

ŽIVOTOPIS

Kristian Šaravanja rođen je 24.09.1992. u Vinkovcima. Pohađao je osnovnu školu „Antun Gustav Matoš “ u Vinkovcima. Nakon osnovne škole upisao je „Gimnaziju Matije Antuna Reljkovića“ u Vinkovcima. Trenutno studira na „Fakultetu elektrotehnike, računarstva i informacijskih tehnologija“ u sklopu Sveučilišta Josipa Jurja Strossmayera u Osijeku, stručni studij – smjer informatika. Odradio je 5 mjeseci stručne prakse u tvrtci Cobe u Osijeku kao iOS developer.