iOS aplikacija za ribolov

Mecanović, Benjamin

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

Sveučilišni studij

IOS APLIKACIJA ZA RIBOLOV „iFish“

Završni rad

Benjamin Mecanović

Osijek, 2019.

Sadržaj

1. UVOD

Ovaj završni rad poslužit će ribolovcima, starim ili onima koji to žele postati, kao važan izvor informacija o vrstama riba i mjestima u regiji za ribolov. Aplikacija se sastoji od dva glavna dijela, mrežnog i izvanmrežnog dijela. Mrežni način rada za funkcioniranje zahtjeva korisnikovu priključenost na internet. Omogućava uvid u lokacije svih ribolovaca na mapi koji svoju lokaciju žele dijeliti. Osnovna zamisao te funkcionalnosti je omogućavanje sigurnijeg planiranja odabira lokacije prije samog odlaska u ribolov. Korisnik u bilo kojem trenutku može prekinuti dijeljenje lokacije. Izvanmrežni način rada zamišljen je kao informativni dio aplikacije. Unutar njega korisnik može pronaći osnovne informacije vezane za ribe kao i lokacije i opise nekih od popularnih ribolovišta na području Slavonije i Baranje.

Aplikacija je razvijena za IOS operacijski sustav korištenjem Xcode integriranog razvojnog okruženja. Kao programski jezik korišten je Swift koji spada u kategoriju jezika za podržavanje više paradigmi. Backend aplikacije ostvaren je korištenjem Firebase baze podataka u stvarnom vremenu dok je za registraciju i prijavu korisnika korištena Firebase Authentication usluga. Za instalaciju vanjskih ovisnosti u projekt korišten je Cocoapods upravitelj ovisnostima.

U prvom poglavlju opisuje se što aplikacija treba raditi. Drugo poglavlje sadrži korištene tehnologije potrebne za izradu aplikacije. Slijedi treće poglavlje unutar kojeg su navedeni sami postupci izrade aplikacije. Zadnje poglavlje predstavlja zaključak u kojem je opisana svrha aplikacije.

1.1. Zadatak završnog rada

Zadatak ovog završnog rada je izraditi IOS aplikaciju koja ribolovcima može poslužiti kao izvor važnih informacija o vrstama riba i mjestima za ribolov na području Slavonije i Baranje. Također omogućava uvid u trenutne lokacije ribolovaca koji svoju lokaciju odluče dijeliti.

2. KORIŠTENE TEHNOLOGIJE

Unutar ovog poglavlja opisane su korištene tehnologije pri izradi aplikacije. Detaljno će se objasniti: operacijski sustav IOS, Xcode IDE, programski jezik Swift, Cocoapods i Firebase.

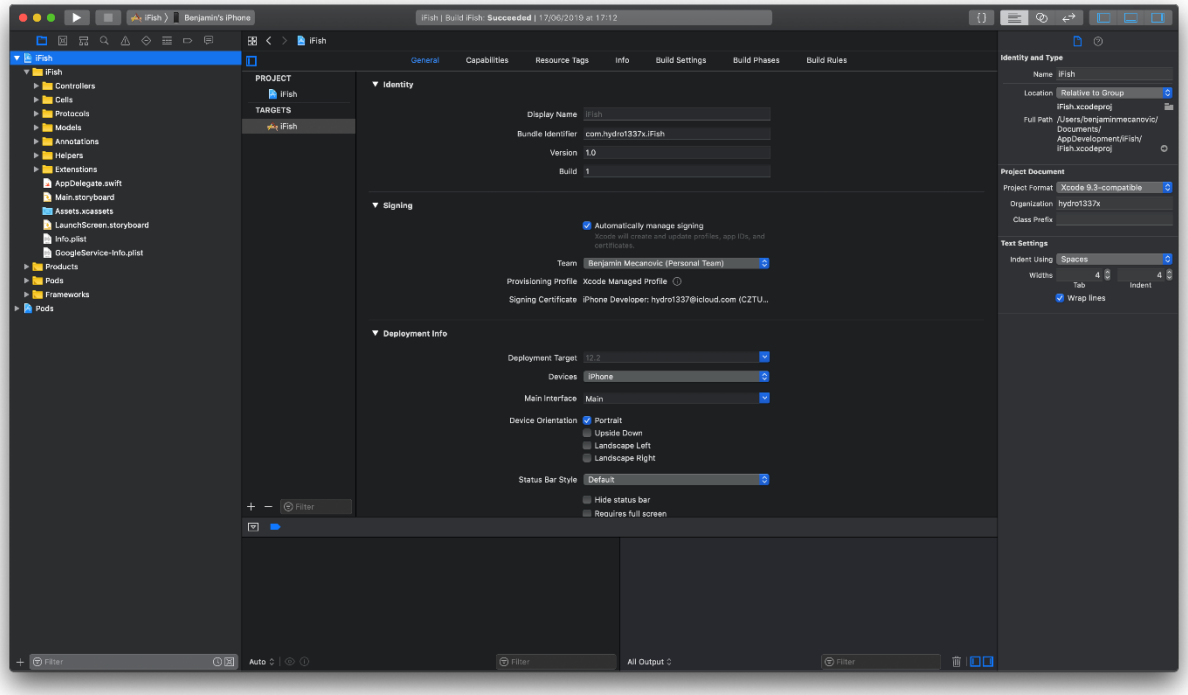
2.1. Operacijski sustav IOS

IOS prethodno znan kao iPhone OS je mobilni operacijski sustav koji je dizajnirao i razvio Apple Inc. isključivo za svoje sklopovlje. To je operacijski sustav koji trenutno pokreće većinu njihovih mobilnih uređaja, uključujući iPhone, iPad i iPod Touch. Trenutno je IOS drugi najpopularniji mobilni operativni sustav nakon operacijskog sustava Android. Korisničko sučelje operacijskog sustava IOS temelji se na direktnoj manipulaciji, korištenju *multi-touch*[1] gesti. Sučelje se sastoji od klizača, prekidača i gumbova. Interakcija s operacijskim sustavom se sastoji od *swipe*[2], *tap*[3], *pinch*[4] i *reverse pinch*[5] gesti koje imaju specifične uloge u kontekstu upravljanja operacijskim sustavom i sučeljem. Unutarnji akcelerometri se koriste kod nekih aplikacija kako bi uređaj mogao reagirati na promjene položaja što omogućava promjenu orijentacije same aplikacije. Operacijski sustav IOS izvedenica je operacijskog sustava macOS pa je time sličan Unix operacijskim sustavima. Postoje četiri sloja apstrakcije unutar operacijskog sustava IOS. Core OS sloj pruža značajke niske razine kao i okvire (engl. *framework*) za sigurnost i interakciju s vanjskim sklopovljem. Core Services sloj pruža usluge koje zahtijevaju gornji slojevi. Media sloj pruža potrebne tehnologije za grafiku, zvuk i video. Cocoa Touch sloj sadržava okvire koji se često koriste pri kreiranju aplikacija.

2.2. Xcode IDE

Xcode je IDE odnosno integrirano razvojno okruženje koje je razvio Apple za potrebe razvoja programske podrške za macOS, IOS, watchOS i tvOS. To je jedini službeno podržani alat za

[1] Tehnologija koja omogućava zaslonu registriranje više dodira u istom trenutku
[2] Gesta povlačenja
[3] Gesta dodira
[4] Gesta širenja
[5] Gesta skupljanja



Slika 2.2. Sučelje Xcode razvojnoj okruženja

razvijanje i objavljivanje aplikacija na App Store. Xcode sadrži sve alate potrebne za stvaranje aplikacije unutar jednog programskog paketa. Sadržava uređivač teksta, provoditelja i sustav za izgradnju što omogućava pisanje, kompiliranje i ispravljanje aplikacije. Kao uređivač koda, Xcode podržava veliki broj programskih jezika: C, C++, Objective – C, Java, AppleScript, Python, Ruby, ResEdit i Swift. Koristi Cocoa, Carbon i Java programske modele. Xcode je osmišljen kako bi razvojnom programeru dao jedan prozor u kojem će raditi. Ima provjeru izvornog koda i značajku automatskog dovršavanja, što će učiniti pisanje izvornog koda mnogo lakšim.

2.3. Programski jezik Swift

Swift je programski kompilirani programski jezik opće namjene koji podržava više paradigmi. Razvila ga je tvrtka Apple Inc. za IOS, macOS, watchOS, tvOS, Linux i z/OS. Swift je dizajniran za rad s Cocoa i Cocoa Touch okvirima i velikim dijelom postojećeg Objective – C koda pisanog za Apple proizvode. Izrađen je s LLVM okvirom prevoditelja otvorenog koda i uključen je u Xcode od verzije 6. izdanje 2014. godine. Na Apple platformama koristi Objective – C runtime biblioteku koja omogućava pokretanje C, Objective – C, C++ i Swift koda unutar jednog

programa. Apple je namijenio da Swift podrži brojne temeljne koncepte povezane s Objective – C jezikom, osobito *dynamic dispatch*[6], *widespread late binding*[7], proširivo programiranje i slične značajke, ali na sigurniji način, olakšavajući pri tome hvatanje grešaka. Swift ima značajke rješavanja nekih uobičajenih programskih pogrešaka kao sto je *null pointer dereferencing*[8] i daje sintaktički šećer kako bi se izbjegla pojava *pyramid of doom*[9]. Swift podržava koncept proširivosti protokola, sustav proširenja koji se može primijeniti na tipove (engl. *types*), strukture (engl. *structures*) i klase (engl. *classes*), koje Apple promiče kao stvarnu promjenu u paradigmama programiranja koju nazivaju „protokolno orijentirano programiranje“.

```
1  import UIKit
2
3  var myAge: Int = 22
4  let myName: String = "Benjamin"
5
6  func printUserInformation(age: Int, name: String) {
7      print("My name is \(name) and I am \(age) years old.")
8  }
9
10 printUserInformation(age: myAge, name: myName)
```

```
My name is Benjamin and I am 22 years old.
```

Slika 2.3. Primjer Swift sintakse

2.4. Cocoapods

CocoaPods je upravitelj ovisnostima na razini aplikacije za Objective – C, Swift i bilo koji drugi jezik koji se izvodi na Objective – C *runtime*[10], kao što je RubyMotion, koji pruža standardni format za upravljanje vanjskim bibliotekama. Razvili su ga Eloy Durán i Fabio Pelosin, koji

[6] Odabir polimorfne metode ili funkcije za vrijeme izvođenja programa
[7] Mehanizam pri kojem se metode pozivane putem objekta ili funkcije pozivane s argumentima traže poimence prilikom izvođenja
[8] Pokazivanje pokazivača na adresu čija je vrijednost nepostojeća odnosno null
[9] Problem koji nastaje prilikom ugnježđivanja uvlačenja za kontrolu pristupa funkciji
[10] Vrijeme izvođenja programa

nastavljaju upravljati projektom uz pomoć i doprinos mnogih drugih. Počeli su s razvojen u kolovozu 2011. i prvi puta objavili 1. rujna 2011. CocoaPods je snažno inspiriran kombinacijom Ruby, RubyGems i Bundler projektima. CocoaPods se usredotočuje na distribuciju koda treće strane i automatsku integraciju u Xcode projekte. Pokreće se iz terminala i također je integriran u JetBrains AppCode razvojno okruženje. Instalira ovisnosti, kao što su biblioteke, tako što se navedu željene ovisnosti što otklanja potrebu ručnog kopiranja izvornih datoteka.

2.5. Firebase

Firebase je platforma za razvoj mobilnih i web aplikacija koju je 2011. razvija tvrtka Firebase Inc., a zatim je tvrtku 2014. kupio Google. Firebase nudi brojne servise kao što su autentifikacija, baza podataka u stvarnom vremenu i brojne alate za praćenje analitike, izvještavanje i popravljanje rušenja aplikacije i stvaranje marketinških i proizvodnih eksperimenata.

2.5.1. Firebase Auth

Firebase Auth je usluga koja omogućava autentifikaciju korisnika koristeći samo kod na klijentovoj strani. Podržava prijavu putem društvenih mreža kao što su Facebook, GitHub, Twitter i Google. Osim toga, uključuje i sustav za upravljanje korisnicima kojim programeri mogu omogućiti provjeru autentičnosti korisnika putem e – pošte i zaporke koji su spremljeni u Firebase.

2.5.2. Firebase Realtime Database

Firebase pruža bazu podataka u stvarnom vremenu i *back end* [11]kao uslugu. Servis pruža programerima API (engl. *application programming interface*[12]) koji omogućava sinkronizaciju podataka o aplikacijama između klijenata i pohranjuje ih u Firebase oblak. Tvrtka nudi klijentske biblioteke koje omogućavaju integraciju s Android, IOS, JavaScript, Java, Objective – C, Swift i Node.js aplikacijama. Bazi podataka se također može pristupiti putem REST API – ja i vezama za

[11] Sloj programske podrške koji omogućava pristup podacima
[12] Aplikacijsko programsko sučelje

nekoliko JavaScript okvira kao što su AngularJS, React, Ember.js i Backbone.js. REST API koristi Server-Sent Events protokol, to je protokol za stvaranje HTTP (engl. *Hyper Text Transfer Protocol*) veza za primanje push obavijesti[13] (engl. *push notifications*) poslužitelja. Programeri koji koriste bazu podataka u stvarnom vremenu mogu osigurati svoje podatke korištenjem sigurnosnih pravila tvrtke na strani poslužitelja.

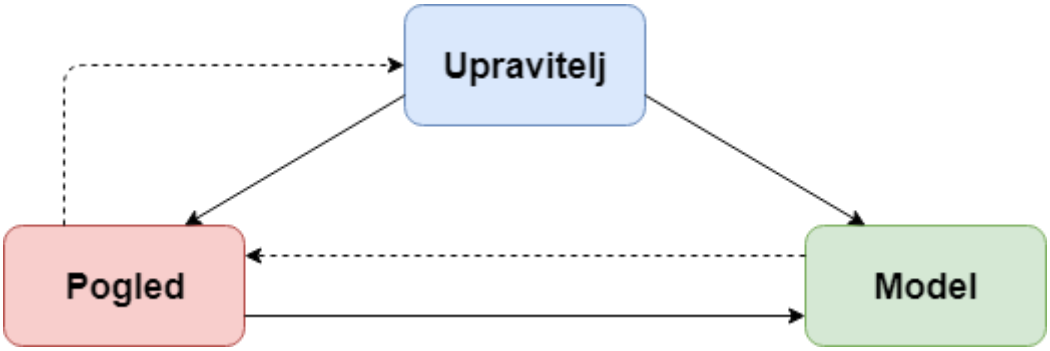
[13] Obavijesti koje aplikacija šalje uređaju na kojem se nalazi

3. RAZVOJ APLIKACIJE

U ovom poglavlju objasnjen je razvoj aplikacije i njezinih dijelova. Na početku se pojašnjava arhitektura zajedno s njezinim dijelovima na temelju koje je aplikacija izrađena te instalacija ovisnosti putem Cocoapods. Zatim se detaljno objašnjava put izrade mrežnog i izvanmrežnog dijela aplikacije.

3.1. MVC arhitektura

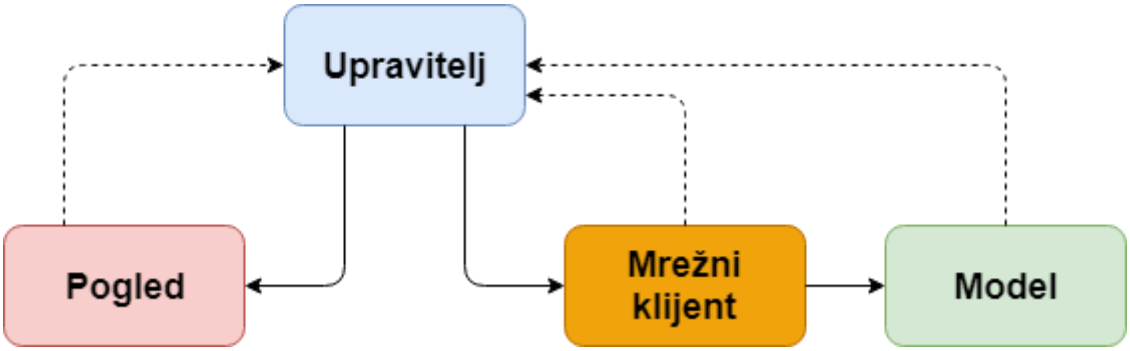
MVC odnosno Model – View – Controller je obrazac arhitekture programske podrške. Koristi se u inženjeringu programske podrške za odvajanje pojedinih dijelova aplikacije u komponente ovisno o njihovoj namjeni. Model se sastoji od podataka, poslovnih pravila, logike i funkcija ugrađenih u poslovnu logiku (engl. *business logic*). Pogled (engl. *view*) je bilo kakav prikaz podataka kao što je obrazac, tablica ili dijagram. U slučaju razvoja mobilnih aplikacija pogled predstavlja sve što omogućava korisniku primanje i slanje informacija samom uređaju odnosno aplikaciji koju koristi. Moguć je prikaz podataka kroz više različitih pogleda. Upravitelj (engl. *controller*) prihvaća ulazne naputke (engl. *inputs*) i pretvara ih u naloge modelu ili pogledu. Ovakva arhitektura olakšava nezavisan razvoj, testiranje i održavanje određene aplikacije.



Slika 3.1 Konceptualni prikaz MVC arhitekture

3.1.1. MVC – N arhitektura

MVC – N odnosno Model – View – Controller – Network nasljeđuje sve značajke MVC arhitekture. Za razliku od MVC arhitekture koja je podijeljena na tri osnovna dijela, MVC – N ima dodatni sloj što omogućava još bolje strukturiranje samog projekta, a uz to dodatno olakšava održavanje koda. Dodatni sloj predstavlja mrežni klijent (engl. *network client*) unutar kojeg je implementirana sva logika za mrežni rad. Mrežni klijent je zadužen za obavljanje mrežnih zahtjeva, serijalizaciju dobivenih podataka sa strane servera u modele definirane unutar projekta i vraćanje podataka samom upravitelju.



Slika 3.1.1. Konceptualni prikaz MVC – N arhitekture

3.2. Instalacija ovisnosti putem CocoaPods (engl. *Dependency injection*)

Nakon kreiranja samog projekta, za potrebe realizacije aplikacije nužno je instalirati ovisnosti kako bi se unutar projekta mogle koristiti metode za korištenje Firebase Auth servisa i manipulaciju bazom podataka u stvarnom vremenu. U projektu je korištena i SVProgressHUD vanjska biblioteka koja u sebi sadržava definicije indikatora aktivnosti te se poziva prilikom odrađivanja vremenski zahtjevnih akcija.

Za instalaciju ovisnosti putem CocoaPods upravitelja ovisnostima koristi se ugrađeni terminal unutar macOS operacijskog sustava. Budući da macOS spada u kategoriju Unix operacijskih sustava možemo koristiti Unix nardbe za navigaciju unutar terminala. Nakon otvaranja terminala naredbom -*cd*, koja služi za promjenu trenutno promatranog direktorija (engl. *change directory*), potrebno je premjestiti se unutar direktorija u kojem je kreiran sami projekt.

```
benjamins-mbp:~ benjaminmecanovic$ cd Desktop/TestProject
```

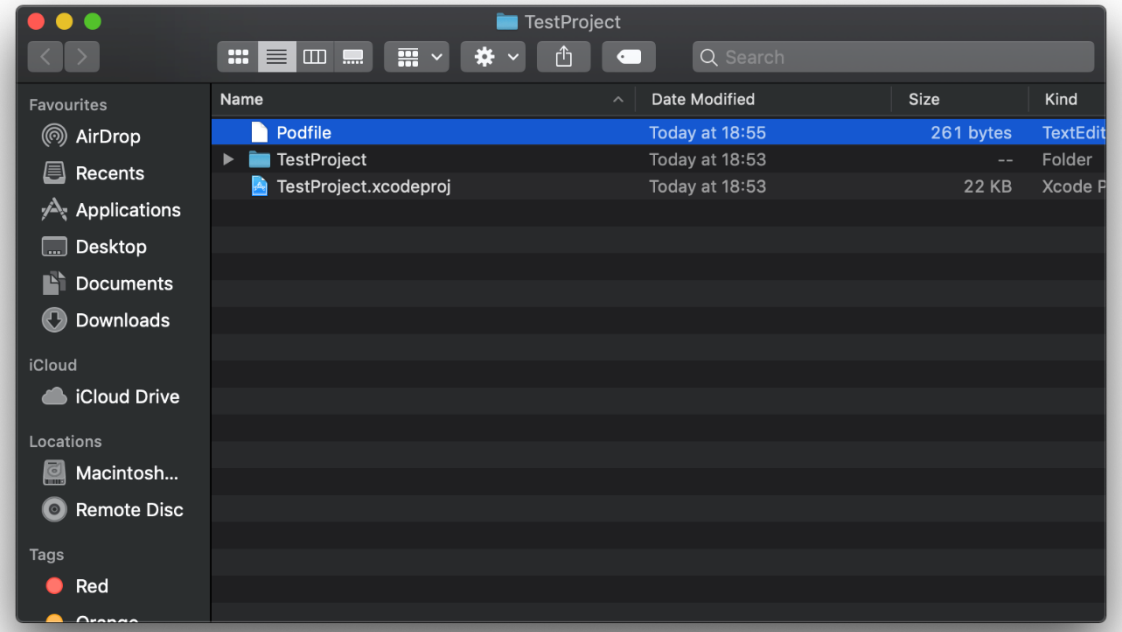
Slika 3.2.1 Naredba za promjenu direktorija

Nakon što smo se premjestili u direktorij samog projekta, potrebno je inicializirati *Podfile* unutar kojeg se navode sve ovisnosti koje želimo instalirati. To činimo naredbom *pod init* .

```
benjamins-mbp:TestProject benjaminmecanovic$ pod init
```

Slika 3.2.2. Naredba za inicializaciju Podfile datoteke

Ako je inicijalizacija prošla uspješno direktorij s projektom trebao bi u sebi sadržavati prazan Podfile, kao što je prikazano na slici 3.2.3.



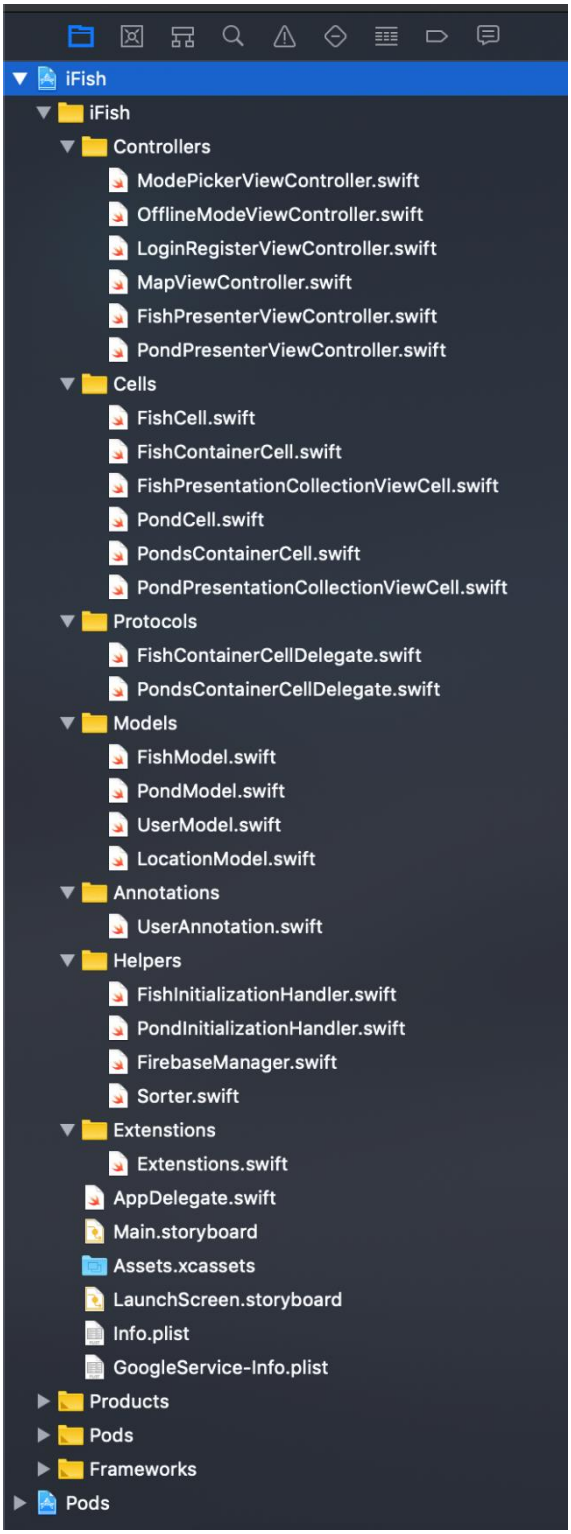
Slika 3.2.3. Prikaz projektnog direktorija s praznom Podfile datotekom

Novo napravljeni Podfile potrebno je otvoriti uređivačem teksta po želji te unutar njega definirate sve potrebne ovisnosti koje se žele koristiti unutar projekta.

Slika 3.2.4. Podfile s navedenim ovisnostima

Zadnji korak koji je potrebno učiniti kako bi se navedene ovisnosti instalirale je pokrenuti samu instalaciju putem naredbe *pod install* . Ukoliko je instalacija protekla uspješno, navedene ovisnosti moguće je koristiti unutar projekta.

3.3. Struktura projekta



Slika 3.2.5. Prikaz strukture projekta

Struktura projekta sastoji se od sedam osnovnih grupa, a to su : upravitelji, ćelije (engl. *cells*), protokoli (engl. *protocols*), modeli (engl. *models*), anotacija (engl. *annotations*), pomoćnih klasa (engl. *helpers*) i proširenja (engl. *extenstions*). Upravitelji su zaslužni za prikaz sadržaja na zaslonu, unutar njih je sadržana logika koja omogućava korisničku interakciju s aplikacijom. Ćelije predstavljaju posebne kontejnere za prikaz sadržaja unutar *CollectionView*[14] elemenata koji su definirani unutar prezentacijskih upravitelja. Protokoli unutar sebe imaju metode koje omogućavaju komunikaciju i prijenos informacija između različitih upravitelja. Modeli predstavljaju specifično definirane tipove podataka koji se koriste unutar aplikacije, u ovom slučaju definirani su razredima, ali mogu biti i strukture. Grupa anotacija sadrži posebno definiranu anotaciju koja omogućava implementaciju posebnog izgleda za prikaz indikatora lokacije korisnika aplikacije. Pomoćne klase sadržavaju metode koje ne bi trebali biti sadržane unutar upravitelja kako se ne bi narušila MVC – N arhitektura, također sadržavaju i FirebaseManager razred koji predstavlja N dio MVC – N arhitekture odnosno mrežni klijent koji je zaslužan za rukovanje Firebase servisom za autentifikaciju i manipulaciju baze podataka stvarnog vremena. Proširenja predstavljaju nadogradnje već postojećih razreda radi lakšeg i jednostavnijeg rukovanja pojedinim metodama te zbog manje kompleksnosti samog koda.

3.4. Razvoj izvanmrežnog dijela aplikacije

Prilikom pokretanja aplikacije prvo se prikazuje ModePickerViewController, to je upravitelj unutar kojeg je definiran izbornik za odlazak u izvanmrežni ili mrežni način rada. Prilikom prvog pokretanja aplikacije nakon same instalacije, prikazat će se uvodna poruka koja opisuje svrhu aplikacije i njene mogućnosti. Uvodna poruka bit će prikazana samo jednom, isključivo pri prvom pokretanju aplikacije. Ta funkcionalnost ostvarena je putem zastavice *isAlertDisplayed* i metode *displayIntroductionAlert* koja mijenja logičko stanje ovisno o pokretanju aplikacije i prikazuje samu poruku.

```
13      var isAlertDisplayed: Bool = false
```

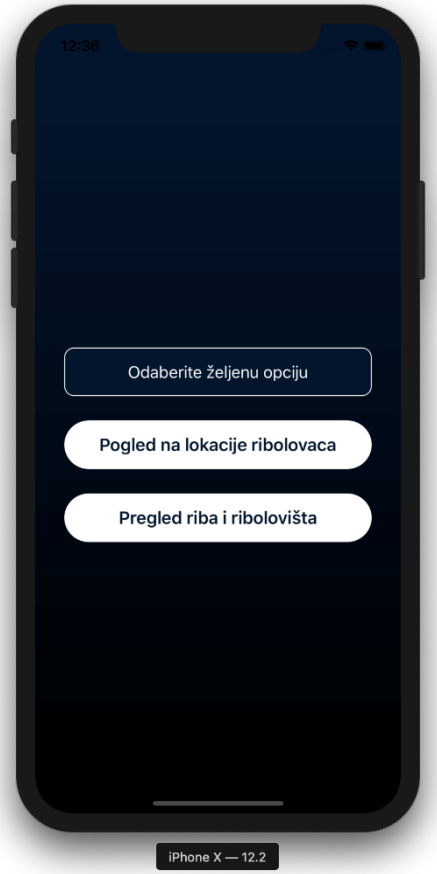
Slika 3.4.1. isAlertDisplayed zastavica

[14] Interaktivni element koji kao osnovne mogućnosti pruža prikaz podataka unutar ćelija i straničenje

```swift
func displayIntroductionAlert() {

    let defaults = UserDefaults.standard
    isAlertDisplayed = defaults.bool(forKey: key)

    if isAlertDisplayed == false {
        let alert = UIAlertController(title: "Dobrodošli na iFish", message: "Aplikacija iFish poslužit će Vam kao vodič za ribolov na području Slavonije i Baranje.
            \n\nImate mogućnost birati između dvije opcije:\nPrva opcija omogućava uvid u Vašu trenutnu lokaciju kao i lokacije ribolovaca koji dijele svoj
            položaj.\nDruga opcija nudi informativni sadržaj vezan za ribolovišta i ribe na našem prostoru.", preferredStyle: .alert)
        alert.view.tintColor = UIColor.CustomColors.midnightBlue
        let action = UIAlertAction(title: "U redu", style: .cancel, handler: nil)
        alert.addAction(action)
        present(alert, animated: true, completion: nil)
        defaults.set(true, forKey: key)
    }
}
```

Slika 3.4.2. Metoda displayIntroductionAlert

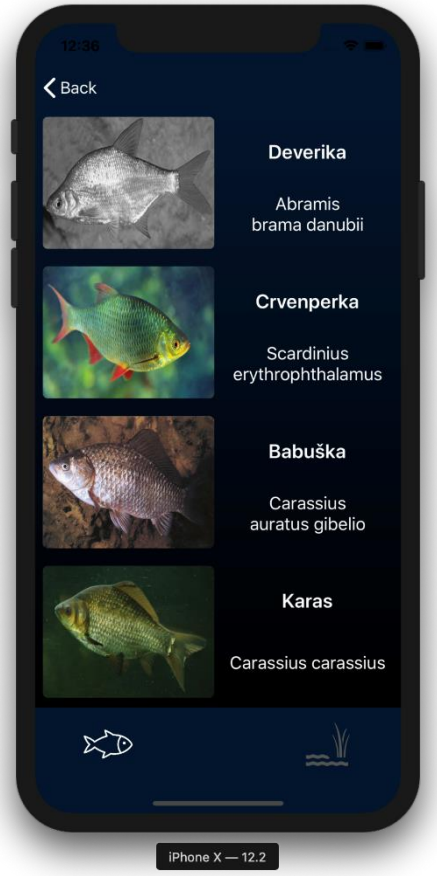
Slika 3.4.3. Prikaz uvodne poruke na zaslonu



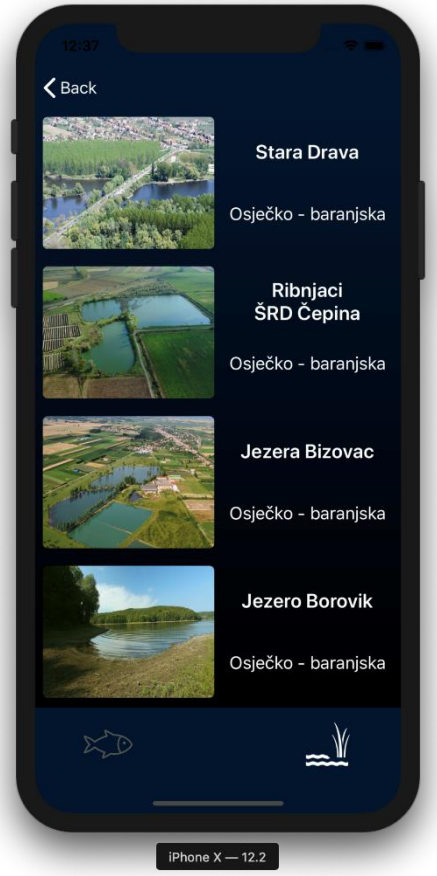
Slika 3.4.4. Prikaz početnog izbonika

Odabirom opcije „Pregled riba i ribolovišta“ korisnik je preusmjeren na *OfflineModeViewController* upravitelja. Novootvoreni upravitelj zaslužan je za navigaciju između vrsta riba i ribolovišta. Unutar njega je definiran collectionView s horizontalnim načinom pomicanja elemenata. U ovom primjeru collectionView se sastoji od dvije ćelije, lijevu predstavlja

FishContainerCell koja je zadužena za prikaz riba, dok desnu stranu predstavlja *PondsContainerCell* koja je zadužena za prikaz ribolovišta. Za prelazak između ćelija koriste se *swipe* akcije za lijevi i desni smjer, ali je moguće i samim pritiskom na gumbove definirane u podnožju.



Slika 3.4.5. CollectionView sa selektiranom FishContainerCell ćelijom



Slika 3.4.6. CollectionView sa selektiranom PondsContainerCell ćelijom

FishContainerCell i PondsContainerCell ćelije predstavljaju kontejnerske ćelije koje u sebi sadržavaju posebno definirane collectionView elemente. Unutar FishContainerCell ćelije nalazi se collectionView sa vertikalnim načinom pomicanja ćelija te koristi FishCell ćelije za prikaz samih riba (Slika 3.4.5.) . Isto je primijenjeno i na PondsContainerCell. FishCell ćelija sastoji se od naziva ribe, latinskog naziva i slike. Ćelije popunjava collectionView koji ju koristi uzimajući

podatke iz polja FishModel elemenata koje se inicijalizira putem pomoćnog razreda FishInitializationHandler. Taj razred predstavljen je kao *singleton*, odnosno instancu tog razreda moguće je pristupiti bilo gdje unutar projekta. Popunjavanje polja s FishModel elementima ostvareno je putem dvije metode sadržane unutar FishInitializationHandler razreda. Metoda createFishFromParameters služi kako bi se kreirala pojedina instanca FishModel elementa, dok metoda fillArray na temelju kreiranih FishModel elemenata stvara i vraća FishModel polje. PondCell ćelija sastoji se od naziva ribolovišta, naziva županije i slike dok je logika za prikaz i popunjavanje ista kao i za FishCell ćeliju koristeći odgovarajući model i metode za PondCell.

```
lazy var collectionView: UICollectionView = {

    let layout = UICollectionViewFlowLayout()
    let cv = UICollectionView(frame: .zero, collectionViewLayout: layout)
    cv.translatesAutoresizingMaskIntoConstraints = false
    cv.backgroundColor = .clear
    layout.minimumLineSpacing = 2
    cv.delegate = self
    cv.dataSource = self
    return cv

}()
```

Slika 3.4.7. Prikaz definicije collectionView elementa

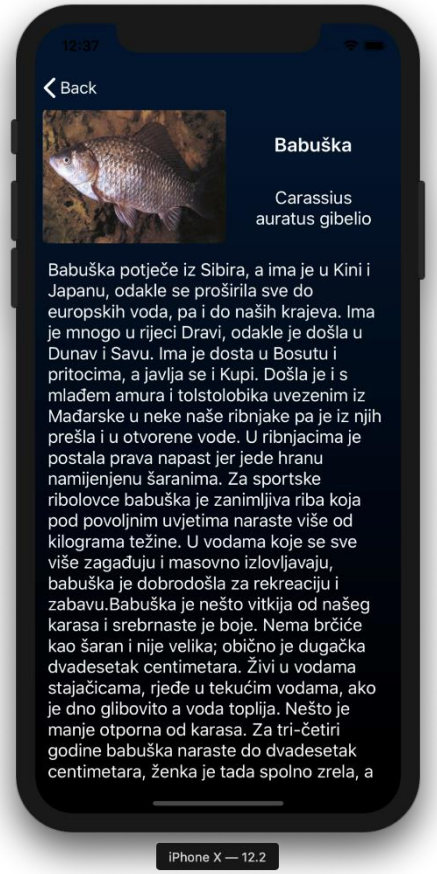
```
class FishModel {

    var name: String
    var latinName: String
    var picture: UIImage
    var description: String

    init(name: String, latinName: String, picture: UIImage, description: String) {
        self.name = name
        self.latinName = latinName
        self.picture = picture
        self.description = description
```

Slika 3.4.8. FishModel razred

Prilikom pritiska na FishCell ćeliju unutar collectionView elementa korisnik se preusmjerava na FishPresentationViewController upravitelj. FishPresentationViewController upravitelj unutar sebe ima definiran novi collectionView s horizontalnim načinom upravljanja. Koristi FishPresentationCollectionViewCell ćelije koje omogućavaju detaljniji prikaz riba tako što za razliku od FishCell elementa koji se sastoji od naziva ribe, latinskog naziva i slike dodatno ima još i opis za pojedinu ribu. Osim toga svaka FishPresentationCollectionViewCell ćelija razvućena je preko cijelog ekrana. Pomoću *swipe* akcija odnosno gesti moguć je prelazak između dostupnih vrsta riba. Ista funkcionalnost je ostvarena i prilikom pritiska na PondCell ćelije.

```
func fillArray() -> [FishModel] {
    var fishArray: [FishModel] = []

    for i in 0..<namesArray.count {
        let name = namesArray[i]
        let latinName = latinNamesArray[i]
        let pictureName = pictureNamesArray[i]
        let description = descriptionsArray[i]
        let fish = createFishFromParameters(name: name, latinName: latinName, pictureName: pictureName, description: description)
        fishArray.append(fish)
    }

    return fishArray

}

func createFishFromParameters(name: String, latinName: String, pictureName: String, description: String) -> FishModel {
    return FishModel(name: name, latinName: latinName, picture: UIImage(named: pictureName) ?? UIImage(), description: description)
}
```

Slika 3.4.9. FishInitializationHandler metode za kriranje i popunjavanje FishModel polja

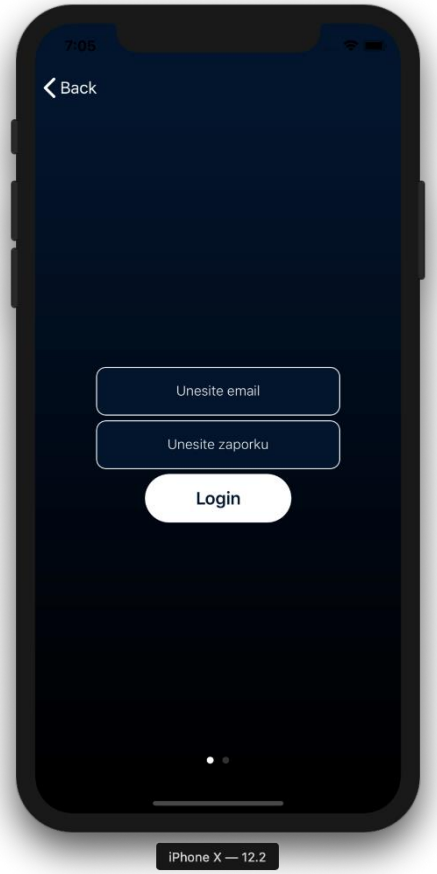


Slika 3.4.10.
FishPresentationCollectionViewCell ćelija

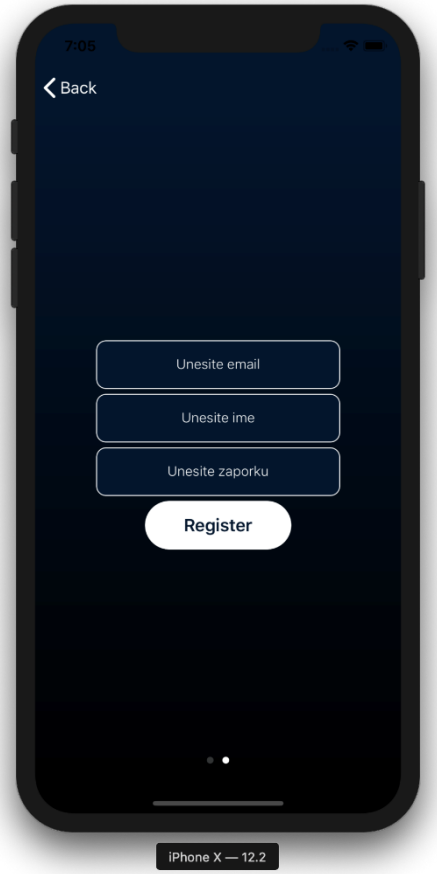
Slika 3.4.11.
PondPresentationCollectionViewCell ćelija

3.5. Razvoj mrežnog dijela aplikacije

Vraćanjem na početni izbornik te odabirom opcije „Pogled na lokacije ribolovaca“ odlazi se na LoginRegisterViewController upravitelj. Unutar LoginRegisterViewController upravitelja smješteni su elementi korisničkog sučelja potrebni kako bi se korisnik mogao registrirati odnosno prijaviti unutar aplikacije. Elementi su smješteni na scrollView elementu. ScrollView je interaktivni element koji omogućava straničenje (engl. *paging*) unutar korisničkog sučelja. U ovom projektu scrollView sadrži dvije stranice od kojih jedna sadržava elemente za prijavu dok druga sadržava elemente za registraciju korisnika.



Slika 3.5.1. Prva stranica scrollView elementa namijenjene prijavi korisnika



Slika 3.5.2. Druga stranica scrollView elementa namijenjena registraciji korisnika

Prva stranica scrollView elementa unutar sebe sadržava dva textField elementa za unos teksta kao i jedan gumb za prijavu (engl. *login*) . Unutar definicije Login gumba postavljena je akcija koja prilikom pritiska na gumb poziva metodu handleLoginButtonPressed. Navedena metoda zadužena je za proces prijave korisnika. Unutar nje se provjerava da li je korisnik unio sve potrebne podatke te se poziva metoda iz FirebaseManager razreda koja izvršava prijavu korisnika te izvještava da li je prijava bila uspješna ili ne. FirebaseManager razred je singleton i stoga je metode tog razreda moguće pozivati bilo gdje unutar projekta.

```
button.addTarget(self, action: #selector(handleLoginButtonPressed), for: .touchUpInside)
```

Slika 3.5.3 Dodavanje akcije na zadani gumb

```
@objc func handleLoginButtonPressed() {
    guard let email = loginEmailTextField.text, let password = loginPasswordTextField.text, loginEmailTextField.text != "" && loginPasswordTextField.text != "" else
        {SVProgressHUD.showError(withStatus: "Unesite sve potrebne podatke."); return}
    SVProgressHUD.show()
    FirebaseManager.loginUser(email: email, password: password, success: { [weak self] (isSuccessful) -> (Void) in
        guard let welf = self else {return}
        if isSuccessful {
            SVProgressHUD.dismiss()
            welf.navigationController?.show(MapViewController(), sender: self)
        }
    }) { (err) -> (Void) in
        if let error = err {
            SVProgressHUD.dismiss()
            SVProgressHUD.showError(withStatus: error.localizedDescription)
        }
    }
}
```

Slika 3.5.4. Akcija za prijavu korisnika povezana s Login gumbom

```
static func loginUser(email: String, password: String, success: @escaping (Bool) -> (Void), failure: @escaping (Error?) -> (Void)) {

    Auth.auth().signIn(withEmail: email, password: password) { (result, error) in
        if error != nil {
            print("Couldn't login user")
            failure(error)
            return
        }
        //Sucessfully logged in
        success(true)
    }
}
```

Slika 3.5.5. Statička metoda mrežnog klijenta koja vrši prijavu putem Firebase Auth metode

Druga stranica scrollView elementa unutar sebe sadržava tri textField elementa za unos teksta kao i jedan gumb za registraciju (engl. *register*) . Unutar definicije Register gumba postavljena je akcija koja prilikom pritiska na gumb poziva metodu handleRegisterButtonPressed. Navedena metoda zadužena je za proces registracije korisnika. Unutar nje se, kao i kod handleLoginButtonPressed metode, provjerava da li je korisnik unio sve potrebne podatke te se poziva metoda iz FirebaseManager razreda koja izvršava registraciju korisnika te daje povratnu informaciju o uspješnosti registracije.

```
@objc func handleRegisterButtonPressed() {
    guard let name = registerNameTextField.text, let email = registerEmailTextField.text, let password = registerPasswordTextField.text, registerNameTextField.text !=
        "", registerEmailTextField.text != "", registerPasswordTextField.text != "" else {SVProgressHUD.showError(withStatus: "Unesite sve potrebne podatke."); return}
    SVProgressHUD.show()
    FirebaseManager.registerUser(name: name, email: email, password: password, success: { [weak self] (isSuccessful) -> (Void) in
        guard let welf = self else {return}
        if isSuccessful {
            SVProgressHUD.dismiss()
            welf.navigationController?.show(MapViewController(), sender: self)
        }
    }) { (err) -> (Void) in
        if let error = err {
            SVProgressHUD.dismiss()
            SVProgressHUD.showError(withStatus: error.localizedDescription)
        }
    }
}
```

Slika 3.5.6. Akcija za registraciju korisnika povezana s Register gumbom

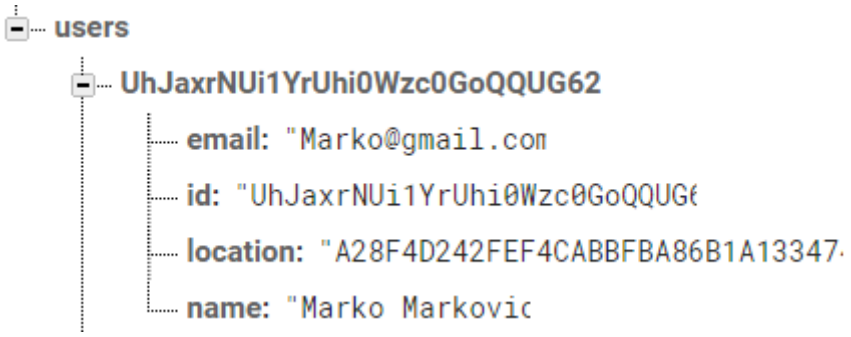
```
static func registerUser(name: String, email: String, password: String, success: @escaping (Bool) -> (Void), failure: @escaping (Error?) -> (Void)) {

    Auth.auth().createUser(withEmail: email, password: password) { (result, error) in
        if error != nil {
            failure(error)
            return
        }
        guard let uid = Auth.auth().currentUser?.uid else {return}
        let locationID = NSUUID().uuidString.replacingOccurrences(of: "-", with: "")
        let values = ["name" : name, "email" : email, "id" : uid, "location" : locationID]
        Database.database().reference().child("users").child(uid).updateChildValues(values, withCompletionBlock: { (error, reference) in
            if error != nil {
                failure(error)
                return
            }
            let locationValues = ["id" : locationID, "lat" : "999", "lon" : "999", "isSharing" : "false"]
            Database.database().reference().child("locations").child(locationID).updateChildValues(locationValues, withCompletionBlock: { (err, ref) in
                if err != nil {
                    failure(err)
                    return
                }
                 success(true)
            })
        })
    }
}
```

Slika 3.5.7. Statička metoda mrežnog klijenta koja vrši registraciju putem Firebase Auth metode

Pritiskom tipke Register korisnik će se registrirati unutar Firebase Authentication servisa uz pretpostavku da su sva polja valjano ispunjena. Unutar Firebase Authentication servisa korisniku se dodjeljuje jedinstveni ključ kokao štjim se razlikuje od ostalih korisnika. Osim toga unutar Firebase baze podataka u stvarnom vremenu stvara se poseban čvor novo registriranog korisnika s njegovim jedinstvenim ključem unutar roditeljskog čvora *users* sa svim njegovim podacima.

Identifier	Providers	Created	Signed In	User UID ↑
marko@gmail.com	✉	Jun 6, 2019	Jun 8, 2019	UhJaxrNUi1YrUhi0Wzc0GoQQUG62

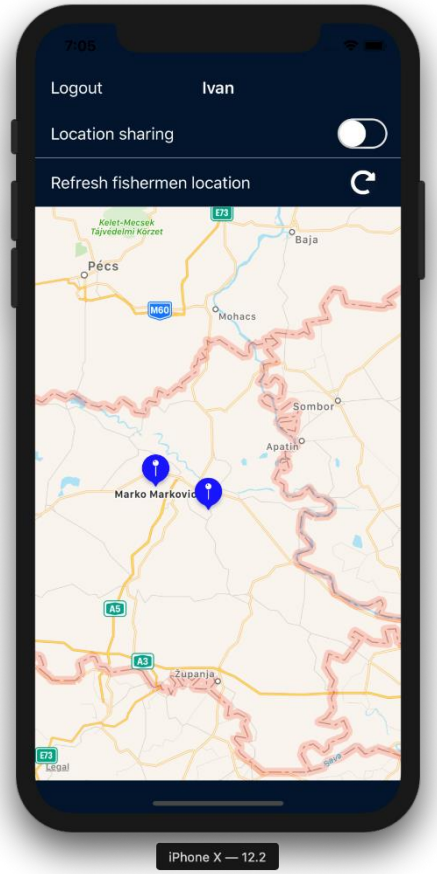
Slika 3.5.8. Prikaz korisnika unutar Firebase Authentication servisa



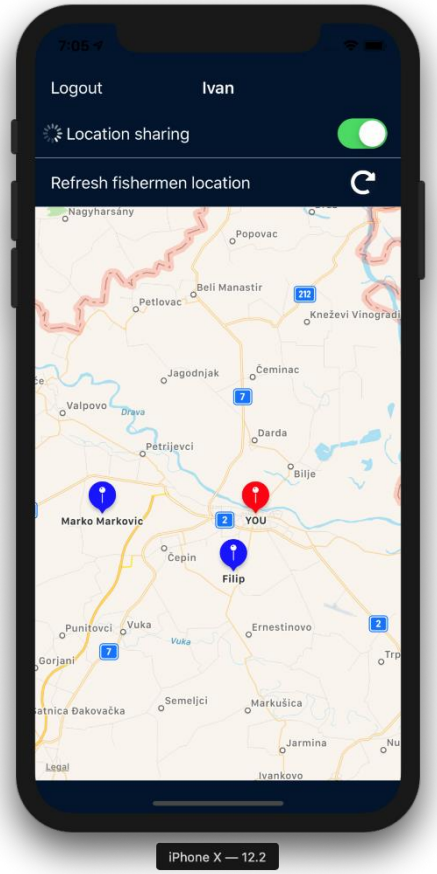
Slika 3.5.9. Čvor registriranog korisnika unutar Firebase baze podataka u stvarnom vremenu

Prema slici 3.5.8. User UID unutar Firebase Authentication servisa podudara se s nazivom čvora registriranog korisnika (Slika 3.5.9.). Jedinstveni ključ se unutar baze podataka koristi kako bi spriječilo pogreške prilikom pretraživanja odnosno kako bi se pojedini korisnici mogli međusobno razlikovati.

Ukoliko je registracija odnosno prijava korisnika bila uspješna korisnik se preusmjerava na MapViewController upravitelj. Unutar MapViewController upravitelja definirani su prekidač za uključivanje ili isključivanje korisničkog dijeljenja lokacije, gumb za osvježavanje trenutnih lokacija svih korisnika koji svoju lokaciju dijele i mapa koja služi za prikaz lokacija korisnika. Uključivanjem prekidača korisnik počinje dijeliti svoju lokaciju odnosno podatke o geografskoj širini (engl. *latitude*) i dužini (engl. *longitude*), za što je zadužena metoda switchChanged koja ovisno o stanju prekidača pokreće ili zaustavlja dijeljenje lokacije putem locationManager menadžera lokacije. Kada locationManager menadžer lokacije pozove metodu startUpdatingLocation poziva se delegatska metoda za ažuriranje podataka lokacije unutar koje se prikupljaju podaci od lokaciji te se putem FirebaseManager uploadLocation metode podaci šalju na bazu podataka u stvarnom vremenu stvarajući pri tome novi čvor unutar roditeljskog čvora *locations.* Pri tome su korisnikov čvor i čvor korisnikove lokacije međusobno povezani na bazi putem jedinstvenog *location* ključa.



Slika 3.5.10. Dijeljenje lokacije - isključeno



Slika 3.5.11. Dijeljenje lokacije - uključeno

```
@objc func switchChanged(locationSwitch: UISwitch) {

    if locationSwitch.isOn {
        FirebaseManager.switchUserLocationSharingStatus(isSharing: true)
        moveLocationSharingLabelRightAndShowActivityIndicator()
        locationManager?.startUpdatingLocation()
    } else {
        FirebaseManager.switchUserLocationSharingStatus(isSharing: false)
        moveLocationSharingLabelLeftAndDismissActivityIndicator()
        locationManager?.stopUpdatingLocation()
    }
}
```

Slika 3.5.12. Metoda promjene stanja prekidača

```swift
func locationManager(_ manager: CLLocationManager, didUpdateLocations locations: [CLLocation]) {
    let location = locations[locations.count-1]
    if location.horizontalAccuracy > 0 {

        let longitude = String(location.coordinate.longitude)
        let latitude = String(location.coordinate.latitude)
        let coordinates = ["lat" : latitude, "lon" : longitude]
        FirebaseManager.uploadLocation(coordinates: coordinates)
    }
}
```

Slika 3.5.13. Delegatska metoda za ažuriranje podataka lokacije

```swift
static func uploadLocation(coordinates: [String : String]) {
    if let uid = getUserUID() {
        getCurrentUserLocationID (uid: uid) { (id) -> (Void) in
            if let locationID = id {
                Database.database().reference().child("locations").child(locationID).updateChildValues(coordinates, withCompletionBlock: { (error, ref) in
                    if let err = error {
                        print(err)
                    }
                })
            }
        }


    } else {
        print("User isnt logged in")
    }
}
```

Slika 3.5.14. Definicija uploadLocation metode za prijenos podataka lokacije na bazu podataka u stvarnom vremenu

Slika 3.5.15. Čvor za lokaciju unutar Firebase baze podataka u stvarnom vremenu

Za prikaz lokacija korisnika zadužena je *updateMapViewWithFetchedUsersAndLocations* metoda. Unutar sebe poziva dvije FirebaseManager metode. Prva koja se poziva je fetchUsers metoda koja kao rezultat vraća UserModel polje svih korisnika unutar *users* roditeljskog čvora

Firebase baze podataka u stvarnom vremenu. Unutar polja se nalaze UserModel objekti koji kao jedan od atributa imaju i locationID. Pozivanjem FirebaseManager fetchLocations metode na temelju locationID atributa iz UserModel polja kao povratnu vrijednost dobijemo LocationModel polje lokacija koje odgovaraju pojedinim korisnicima. Problem predstavlja što indeksi polja lokacija i polja korisnika nisu usklađeni, stoga se još poziva i Sorter metoda matchArrayIndexes koja rješava taj problem. Na kraju dobijemo da i-ti indeks polja korisnika ima svoju lokaciju na i-tom indeksu polja lokacija. Posloženi indeksi nam uvelike olakšavaju dodavanje anotacija korisnika na mapu jer ne moramo prilikom svakog dodavanja ponovno pretraživati koji se korisnik nalazi na kojoj lokaciji.

```
func updateMapViewWithFetchedUsersAndLocations() {  // also sets the navigationBarTitle to the currently logged user's name
    FirebaseManager.fetchUsers { [weak self] (usersArray) -> (Void) in
        guard self != nil else {return}
        FirebaseManager.fetchLocations(usersArray: usersArray, completion: { [weak self] (locationsArray) -> (Void) in
            guard let welf = self else {return}
            welf.usersArray = usersArray
            welf.locationsArray = locationsArray
            welf.locationsArray = Sorter.matchArrayIndexes(usersArray: welf.usersArray, locationsArray: welf.locationsArray)
            for i in 0..<welf.usersArray.count {

                let regionCenter = CLLocationCoordinate2D(latitude: 45.540143, longitude: 18.697616)
                let region = MKCoordinateRegion(center: regionCenter, latitudinalMeters: 100000, longitudinalMeters: 100000)
                welf.mapView.setRegion(region, animated: true)

                if welf.locationsArray[i].lat != 999 && welf.locationsArray[i].lon != 999 {
                    if welf.locationsArray[i].isSharing == true {
                        let lat = welf.locationsArray[i].lat
                        let lon = welf.locationsArray[i].lon

                        if welf.usersArray[i].id == FirebaseManager.getUserUID() {
                            let dropPin = UserAnnotation(id: welf.usersArray[i].id)
                            dropPin.title = "YOU"
                            let dropPinCenter = CLLocationCoordinate2D(latitude: lat , longitude: lon)
                            dropPin.coordinate = dropPinCenter
                            welf.mapView.addAnnotation(dropPin)
                        } else {
                            let dropPin = UserAnnotation(id: welf.usersArray[i].id)
                            dropPin.title = welf.usersArray[i].name
                            dropPin.subtitle = welf.usersArray[i].email
                            let dropPinCenter = CLLocationCoordinate2D(latitude: lat , longitude: lon)
                            dropPin.coordinate = dropPinCenter
                            welf.mapView.addAnnotation(dropPin)
                        }
                    } else {
                        if welf.usersArray[i].id == FirebaseManager.getUserUID() {
                            welf.setupNavigationBarTitle(user: welf.usersArray[i])
                        }
                    }
                }
            }
        })
    }
}
```

Slika 3.5.16. updateMapViewWithFetchedUsersAndLocations metoda za prikaz korisnika na mapi

```
static func fetchUsers(completion: @escaping ([UserModel]) -> (Void)) {
    var usersArray = [UserModel]()
    Database.database().reference().child("users").observeSingleEvent(of: .value) { (snapshot) in
        let numberOfChilds = snapshot.childrenCount
        Database.database().reference().child("users").observe(.childAdded, with: { (snap) in
            if let dictionary = snap.value as? [String : AnyObject] {
                if let name = dictionary["name"] as? String, let email = dictionary["email"] as? String, let id = dictionary["id"] as? String, let locationID =
                    dictionary["location"] as? String {
                    usersArray.append(UserModel(name: name, email: email, id: id, location: locationID))
                      print(usersArray.count)
                    if numberOfChilds == usersArray.count {
                        completion(usersArray)
                    }
                }
            }
        })
    }
}
```

Slika 3.5.17. Definicija fetchUsers FirebaseManager metode za dohvaćanje korisnika s baze podataka u stvarnom vremenu

```
static func fetchLocations(usersArray: [UserModel], completion: @escaping ([LocationModel]) -> (Void)) {
    var locationsArray = [LocationModel]()
    print("numberOfItems: " + "\(usersArray.count)")
    Database.database().reference().child("locations").observe(.childAdded) { (snapshot) in
        if let dictionary = snapshot.value as? [String : AnyObject] {
            if let locationID = dictionary["id"] as? String, let latitude = dictionary["lat"] as? NSString, let longitude = dictionary["lon"] as? NSString, let isSharing
                = dictionary["isSharing"] as? String {
                locationsArray.append(LocationModel(lat: latitude.doubleValue, lon: longitude.doubleValue, id: locationID, isSharing: isSharing))
                if usersArray.count == locationsArray.count {
                    completion(locationsArray)
                }
            }
        }
    }
}
```

Slika 3.5.18. Definicija fetchLocations FirebaseManager metode za dohvaćanje lokacija s baze podataka u stvarnom vremenu

```
static func matchArrayIndexes(usersArray: [UserModel], locationsArray: [LocationModel]) -> [LocationModel] {
    var tempArray = [LocationModel]()
    for _ in 0..<usersArray.count {
        tempArray.append(LocationModel(lat: 888, lon: 888, id: "???", isSharing: "false"))
    }
    for i in 0..<usersArray.count {
        for j in 0..<usersArray.count {
            if usersArray[i].location == locationsArray[j].id {
                tempArray.insert(locationsArray[j], at: i)
            }
        }
    }
    return tempArray
}
```

Slika 3.5.18. Definicija matchArrayIndexes Sorter metode za uparivanje indeksa LocationModel i UserModel polja

Pritiskom na Refresh gumb poziva se odgovarajuća akcija zadužena za osvježavanje lokacije ribolovaca unutar koje se ponovno poziva metoda

updateMapViewWithFetchedUsersAndLocations. Dok se podaci dohvaćaju s baze podataka u stvarnom vremenu koristi se SVProgressHUD indikator za aktivnost koji korisniku daje do znanja da se u pozadini izvršava određeni proces.

4. ZAKLJUČAK

Mobilni su uređaji postali neizostavni dio u svakodnevici života ljudi u današnjem svijetu. Uzimajući u obzir tu činjenicu napravljen je i ovaj završni rad. Vrlo je rijetko da ne možemo pronaći određene informacije koristeći internet ako barem donekle znamo koristiti internet pretraživač. Međutim, problem ne predstavlja dostupnost informacija već vrijeme koje je potrebno prikupiti željene informacije. Aplikacijom završnog rada cilj je bio napraviti vodič za ribolov unutar kojeg bi se nalazile sve važne informacije vezane za ribe i ribolovišta na području Slavonije i Baranje i time uštediti vrijeme tako što bi sve bilo dostupno unutar jedne aplikacije. Aplikacija može biti od koristi novim, ali i iskusnim ribolovcima. Novi ribolovci mogu steći osnovne informacije o ribama i mjestima na kojima je pogodno riboloviti dok iskusniji mogu iskoristiti činjenicu da aplikacija nudi i lokacije njenih korisnika te time saznati da li je njihovo omiljeno mjesto zauzeto. Korištenje aplikacije napravljeno je da bude intuitivno i jednostavno, a moguće je i navigirati na više načina ovisno što korisnik preferira. Budući da je aplikacija temeljena na MVC – N arhitekturi buduće izmjene moguće je napraviti relativno jednostavno. Ta činjenica omogućava i lako provođenje nadogradnji funkcionalnosti ukoliko bude potrebno.

LITERATURA

[1] Wikipedija, IOS, dostupno na:

https://en.wikipedia.org/wiki/IOS (lipanj, 2019.)

[2] Technopedia, IOS, dostupno na:

https://www.techopedia.com/definition/25206/ios (lipanj, 2019.)

[3] Zero To App Store, Što je Xcode IDE, dostupno na:

https://www.zerotoappstore.com/what-is-xcode-and-why-do-i-need-it (lipanj, 2019.)

[4] Wikipedija, Programski jezik Swift, dostupno na:

https://en.wikipedia.org/wiki/Swift_(programming_language) (lipanj, 2019.)

[5] Wikipedija, Cocoapods, dostupno na:

https://en.wikipedia.org/wiki/CocoaPods (lipanj, 2019.)

[6] Wikipedija, Firebase, dostupno na:

https://en.wikipedia.org/wiki/Firebase (lipanj, 2019.)

[7] Firebase, Firebase, dostupno na:

https://firebase.google.com/docs (lipanj, 2019.)

[8] Wikipedija, MVC arhitektura, dostupno na:

https://hr.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller (lipanj, 2019.)

SAŽETAK

U ovom završnom radu razvijena je IOS mobilna aplikacija za ribolov. Aplikacija se sastoji od dva osnovna dijela. Prvi predstavlja izvanmrežni dio te nudi korisniku informacije vezano za ribe i ribolovišta na području Slavonije i Baranje. Drugi predstavlja mrežni dio aplikacije unutar kojeg korisnik stvara svoj račun koristeći Firebase kao backend. Korisnik također ima opcije dijeljenja svoje lokacije kao i uvid u lokacije svih korisnika koji svoju lokaciju dijele. Aplikacija je bazirana na MVC – N arhitekturi što omogućava lako održavanje koda kao i jednostavnost kasnijih nadogradnji funkcionalnosti.

Ključne riječi: IOS, ribolov, Firebase, dijeljenje lokacije, MVC – N arhitektura

ABSTRACT

IOS fishing application

In this bachelor's thesis an IOS mobile application was developed. The application consists of two main parts. The first represents the offline part that offers the user information about fish and fishing areas in Slavonia and Baranja region. The second represents the online part in which a user creates a user profile with Firebase as the backend. The user has also the option to share its location and see the location of other users who share their location. The application is based on the MVC – N architecture, enabling easy code maintenance as well as ease of later upgrades of functionality.

Keywords: IOS, fishing, Firebase, location sharing, MVC – N architecture

ŽIVOTOPIS

Benjamin Mecanović je rođen 14. siječnja 1997. godine u Vinkovcima. Od 2004. do 2012. godine pohađa Osnovnu Školu Zrninskih u Nuštru. Za vrijeme osnovnoškolskog obrazovanja sudjelovao je na natjecanjima iz matematike, povijesti, tjelesno zdravstvene kulture i informatike. Godine 2012. upisuje Opću Gimnaziju Matije Antuna Reljkovića u Vinkovcima. Tijekom srednjoškolskog obrazovanja sudjelovao je na natjecanjima iz biologije, tjelesno zdravstvene kulture i informatike. Godine 2016. završava srednju školu polaganjem ispita državne mature te upisuje Fakultet elektrotehnike, računarstva i informacijskih tehnologija u Osijeku, preddiplomski studij računarstva.