



# Diplomski rad

Prvog ciklusa studija

## Razvoj IOS aplikacije za prikaz radara i stanja na putevima

Student  
Haseljić Eldar

Mentor:  
Dr.sc. Mešković Emir  
red.prof.

Tuzla, 2021.

---

# Sažetak

*U ovom diplomskom radu ćemo govoriti o razvoju IOS aplikacije za prikaz radara i stanja na putevima korištenjem programskog jezika Swift i njenoj upotrebi. Unutar aplikacije za tip geografske karte na kojoj su prikazani radari i stanja na putevima korišten je MapKit. MapKit predstavlja Apple - ov Framework za predstavljanje geografskih karti za iOS, iPadOS, tvOS, watchOS i macOS uređajima. Korisnik pored vizualnog prikaza lokacija gore navedenih elemenata ima mogućnost dodavanja novih, također ima i opciju za filtriranje i brisanja lokacija radara ili stanja na putevima. Podaci su spremljeni u Cloud Firestore bazu podataka, koja je fleksibilna, skalabilna, NoSQL baza podataka koja se koristi prilikom razvoja mobilnih, web aplikacija. Aplikacija je prevedena na tri jezika Njemački, Engleski i Bosanski*

Ključne riječi: Aplikacija, Swift, MapKit, Firestore, Radar, Stanje na putevima

# Abstract

*In this graduate work, we will talk about the development of IOS application for displaying radar and road conditions using the programming language Swift and its use. MapKit was used within the application for the type of map showing radars and road conditions. MapKit is Apple's Framework for presenting geographic maps for iOS, iPadOS, tvOS, watchOS and macOS devices. In addition to visually displaying the locations of the above elements, the user has the option of adding new ones and has the option to filter and delete radar locations or road conditions. The data is stored in a Cloud Firestore database, which is a flexible, scalable, NoSQL database used in the development of mobile, web applications. The application has been translated into three languages: German, English and Bosnian*

Keywords: Application, Swift, MapKit, Firestore, Radar, Road Condition

# Sadržaj

1. Uvod.....	8
2. Korištene tehnologije i alati .....	9
2.1. Swift programski jezik .....	9
2.2. Xcode.....	10
2.2.1. Kreiranje Xcode projekta .....	10
2.3. iOS Simulator .....	13
2.4. CocoaPods .....	13
2.4.1. Povezivanje CocoaPods sa projektom.....	13
2.5. Reaktivno programiranje .....	15
2.5.1. RxSwift i RxCocoa Framework .....	15
2.6. Firebase - Cloud Firestore baza podataka .....	16
2.7. Model – View – ViewModel (MVVM) .....	18
3. Struktura aplikacije .....	19
4. Korisničko okruženje aplikacije.....	22
5. Opis rada i implementacija aplikacije .....	23
5.1. Opis rada glavnog zaslona.....	23
5.1.1. Implementacija glavnog zaslona .....	23
5.2. Opis rada zaslona za prikaz radara .....	25
5.2.1. Implementacija zaslona za prikaz radara.....	26
5.3. Opis rada zaslona za prikaz stanja na putevima .....	33
5.3.1. Implementacija zaslona za prikaz stanja na putevima.....	34
5.4. Opis rada zaslona za filtriranje .....	41
5.4.1. Implementacija zaslona filtriranje .....	42
5.5. Opis rada zaslona za prikaz detalja .....	48
5.5.1. Implementacija zaslona za prikaz detalja .....	51
5.6. Opis rada zaslona za prijavu novih radara ili stanja na putevima .....	60
5.6.1. Implementacija zaslona za prijavu radara ili stanja na putu.....	63
5.7. Unos podataka u firebase uz pomoć skripti.....	70
6. Zaključak.....	70
7. Literatura.....	71

# Popis slika

Slika 1. Prikaz početnog zaslona aplikacije u landscape i portrait modu.....	8
Slika 2. Swift programski jezik .....	9
Slika 3. Xcode razvojna okolina.....	10
Slika 4. Welcome to Xcode .....	10
Slika 5. Xcode interface .....	11
Slika 6. Opcije novog Xcode projekta.....	11
Slika 7. iOS Simulator.....	13
Slika 8. Cloud Firestore.....	16
Slika 9. MVVM (Model – View- ViewModel) arhitektura .....	18
Slika 10. Struktura aplikacije .....	19
Slika 11. Zaslون pokretanja.....	22
Slika 12. Glavni zaslon.....	22
Slika 13. Prikaz ostalih screenova koji su vidljivi u aplikaciji.....	22
Slika 14. Prikaz upozorenja prilikom klika na LinkedIn.....	23
Slika 15. Zaslون sa prikazanim radarima .....	25
Slika 16. Zaslون sa prikazanim radarima, mapa teren.....	25
Slika 17. Pozicije osnovnih dugmadi na mapi .....	33
Slika 18. Prikaz zaslona za prikaz stanja na putevima .....	33
Slika 19. Prikaz funkcionalnosti na zaslonu za prikaz stanja na putevima .....	38
Slika 20. Prikaz svih znakova korištenih za prikaz stanja i radara na mapi.....	39
Slika 21. Zaslون za filtriranje stanja na putevima i zaslون za filtriranje radara .....	41
Slika 22. Grafički prikaz FilterOptionTableViewCell ćelije.....	45
Slika 23. Prikaz malog opsia stanja radara ili stanja na putu .....	48
Slika 24. Prikaz zaslona za prikaz detalja radara, stanja na putevima i općih informacija .....	48
Slika 25. Uspješno prijavljen neakrivan radar ili stanje na putu .....	50
Slika 26. Pozicija oznake za prijavu.....	60
Slika 27. Prikaz report zaslona .....	62

# Kod Reference

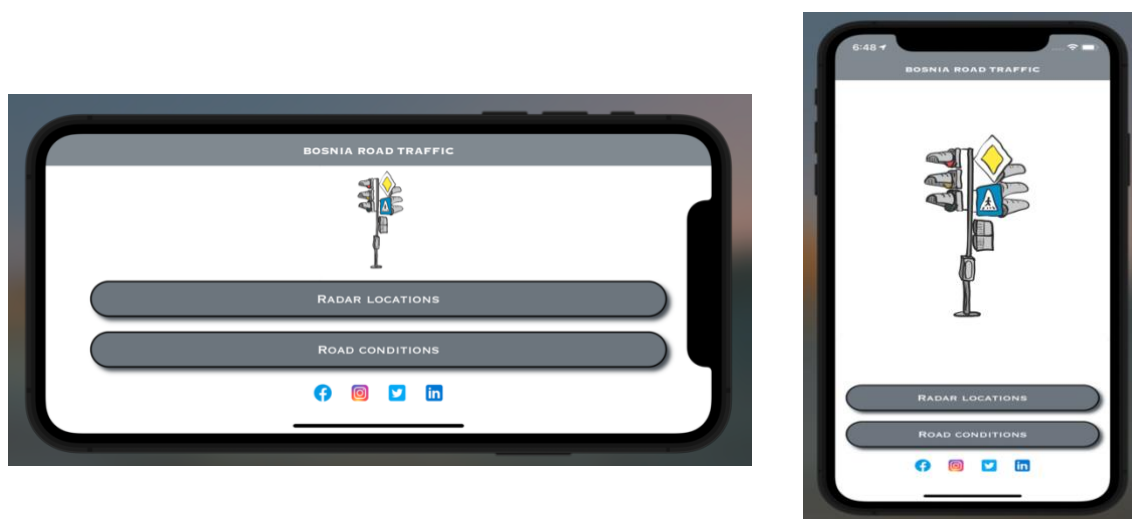
Kod 1. Komanda za instalaciju cocoaPods .....	13
Kod 2. Komanda za inicijalizaciju Podfile datoteke .....	14
Kod 3. Sadržaj Podfile datoteke BosniaRoadTraffic aplikacije .....	14
Kod 4. Komade za instaliranje i nadogradnju biblioteka koje su navedene u Podfile .....	15
Kod 5. Komande za importiranje Rx biblioteka unutar Podfile datoteke .....	16
Kod 6. Komande za importovanje Firebase biblioteka unutar Podfile datoteke .....	17
Kod 7. Konfiguracija Firebase baze unutar projekta .....	17
Kod 8. Sadržaj AppDelegate.swift fajla .....	20
Kod 9. Prikaz sadržaja funkcije pushViewController(viewController: _) .....	24
Kod 10. Reaktivni wrapper za TouchUpInside event nad dugmetom .....	24
Kod 11. Sadržaj funkcije openExternalUrl(urlString:_) .....	24
Kod 12. Prikaz funkcije showRadars() .....	26
Kod 13. Sadržaj checkLocationServices() funkcije .....	26
Kod 14. Sadržaj klase Adviser .....	27
Kod 15. Sadržaj funkcije checkLocationAuthorization() .....	28
Kod 16. Kod za slušanje promjene vrijednosti variable userLocationStatus .....	28
Kod 17. Sadržaj funkcije fetchData() .....	29
Kod 18. Sadržaj MKAnnotation interface-a .....	29
Kod 19. Sadržaj Radars strukture .....	30
Kod 20. RadarMarkerView objekat za konfiguraciju radar oznake na mapi .....	30
Kod 21. Funkcija za dohvaćanje radara sa apia .....	31
Kod 22. Predplatnik unutar viewControllera koji čeka na izmjene radarsArray variable .....	32
Kod 23. showRadars funkcija za prikaz radara na mapi .....	32
Kod 24. Sadržaj prepareMapAndFilter funkcije unutar RadarsViewControllera .....	32
Kod 25. Sadržaj funkcije showRoadConditions() .....	34
Kod 26. Sadržaj funkcije fetchData() unutar RoadConditionsViewModel .....	35
Kod 27. Sadržaj RoadConditionDetails objekta .....	35
Kod 28. Sadržaj reloadScreen funkcije .....	35
Kod 29. Sadržaj funkcije getRoadConditionFullReport .....	36
Kod 30. Sadržaj funkcije getRoadConditions() .....	37
Kod 31. Sadržaj RoadCondition objekta .....	38
Kod 32. Sadržaj RoadConditionMarkerView-a .....	40
Kod 33. Prikaz inicijalizacije FilterScreena .....	42
Kod 34. Sadržaj setData funkcije unutar FilterViewControllera .....	43
Kod 35. Opis tapEditButton funkcije .....	43
Kod 36. Sadržaj funkcije presentViewController .....	43
Kod 37. Dvije glavne funkcije prilikom konfigurisanja tableView komponente .....	43
Kod 38. Sadržaj FilterOptionTableViewCell ćelije .....	44
Kod 39. Sadržaj TypeOption protokola .....	45
Kod 40. Ostali sadržaj FilterViewControllera .....	46
Kod 41. Primjer inicijalizacije prijemnika promjena filtera .....	47
Kod 42. Sadržaj prepareMapAndFilter funkcije unutar RoadConditionViewControllera .....	51
Kod 43. Prikaz funkcija potrebnih za dohvaćanje opštih informacija iz baze mobitela .....	52
Kod 44. Prikaz tapinfoButton funkcije .....	53
Kod 45. Prikaz funkcija za učitavanje DetailsViewCotrollera i učitavanje ćelija na istom .....	54
Kod 46. Prikaz sadržaj ViewProtocol-a .....	54

Kod 47. Sadržaj DetailsViewMoldela.....	55
Kod 48. Sadržaj DetailsViewCell objekta.....	56
Kod 49. Sadržaj tapReportButton funkcije unutar zaslona za detalje .....	57
Kod 50. Sadržaj deleteElement i updateOrDelete funkcija .....	58
Kod 51. Sadržaj updateVisibility funkcije .....	59
Kod 52. Implementacija otvaranja DetailsViewControllera sa oznake .....	59
Kod 53. Sadržaj setReportVisibility funkcije.....	63
Kod 54. Prikaz pretplatnika na dugmad za reporting .....	63
Kod 55. Učitavanje ćelija na UITableView unutar ReportViewControllera .....	65
Kod 56. Sadržaj handleApplyButton unutar ReporRoadConditionCell .....	66
Kod 57. Sadržaj handleApplyButton unutar ReporRoadConditionCell .....	67
Kod 58. Sadržaj addNewRadar i addNewRoadCondition funkcija u viewModel-ima.....	68
Kod 59. Sadržaj addNewRadar i addNewRoadCondition unutar mainManagera .....	69
Kod 60. Načini pokretanja skripti za unos i brisanje podataka .....	70

# 1. Uvod

Aplikacija o kojoj ćemo pričati u ovom diplomskom radu je namijenjena svim korisnicima koji se nalaze u saobraćaju. Ona sadrži prikaz svih radara na putevima, te prikaz samog stanja na putevima. Najveću korist ove aplikacije bi imali vozači motornih vozila, jer bi u bilo kojem trenutku mogli saznati gdje se nalazi radar ili bi mogli provjeriti stanje na putu.

Naravno aplikacija bi bila korisna i za pješake jer bi npr. mogli vidjeti ako je se negdje desila prometna nesreća. Također unutar aplikacije imamo dijelove gdje korisnik može sam da unosi nove pozicije radara ili pozicije gdje se desila neka nesreća, zastoј, moguća poledica ili drugi oblici stanja na putevima, ukoliko ta pozicija nije već navedena.



**Slika 1. Prikaz početnog zaslona aplikacije u landscape i portrait modu**

Ukoliko bi aplikacija bila u vlasništvu neke firme ili ovlaštenog lica te ako isti iz nekog izvora posjeduju veću količinu pozicija gdje se nalaze radari ili neka stanja na putevima ona se mogu unijeti u bazu uz pomoć odgovarajućih skripti. Unutar same aplikacije je ugrađen sistem brisanja pozicija radara ili stanja na putevima nakon određenog broja prijava korisnika da se na tim pozicijama više ne nalazi radar ili neki drugi tip stanja na putu, ali također postoje i odgovarajuće skripte za brisanje istih.

Sama aplikacija je pravljenjena samo za iOS platformu ali je moguće da se ista napravi i za druge mobilne platforme. Aplikacija je prevedena na tri jezika Njemački, Engleski i Bosanski. Na mapu unutar aplikacije Dark mode je primjenjiv tako da ukoliko je ta opcija uključena na mobitelu mapa će biti crne boje i okolne ikonice će se također prilagoditi istom. Aplikaciju je moguće koristiti u vodoravnom (landscape) ili vertikalnom (portrait) modu



## 2. Korištene tehnologije i alati

Krenut ćemo od osnovnih pojmova s kojima ćemo se susretati u ovom diplomskom radu i koji su bili neophodni za razvoj ove iOS aplikacije. Objasnit ćemo za početak pojmove poput Xcode, Swift, Rx biblioteke i druge.

Za izradu ove aplikacije potrebne su nam tri glavne stavke:

- MacOS računalo ili virtualna mašina koja pokreće MacOS
- Xcode
- iOS SDK (eng. software development kit)

### 2.1. Swift programski jezik

Swift je objektno orijentiran programski jezik koji je kreiran od strane Apple-a za razvoj programa i aplikacija na operativnim sistemima iOS i OS X. Prvi put objavljen 2014. godine, te je razvijen kao zamjena za Apple-ov programski jezik, koji je ranije korišten, Objective-C. Swift radi s Apple-ovim Framework-sima Cocoa i Cocoa Touch, a ključni aspekt Swift-ovog dizajna bila je mogućnost interakcije s ogromnim brojem postojećeg Objective-C koda razvijenog za Apple-ove proizvode u prethodnim desetljećima. Swift koristi LLVM (Low Level Virtual Machine) kompajler za kompajliranje koda koji dolazi sa Xcode 6 te koristi Objective-C runtime. Stoga, unutar jednog programa možemo koristiti kod pisan u C, Objective-C, C++ i Swift programskom jeziku.



Slika 2. Swift programski jezik

## 2.2. Xcode

Xcode je razvojna okolina (eng. integrated development environment - IDE) koju koristimo za razvoj iPhone aplikacija. Xcode sadrži source editor, kompajler, emulator, razvojne Framework-e i još puno elemenata koja će nam olakšati razvoj iPhone aplikacija. U principu, od alata za razvoj potreban nam je samo Xcode. Možemo koristiti neki drugi IDE za razvoj, no kako Apple ima striktna pravila kojima regulira strukturu aplikacije koje mogu biti objavljene na App Store-u, preporučljivo je koristiti Xcode jer sam automatski otklanja potencijalne neregularnosti koje bi mogle prouzročiti probleme. Xcode je besplatan te se može skinuti sa App Store-a.



Slika 3. Xcode razvojna okolina

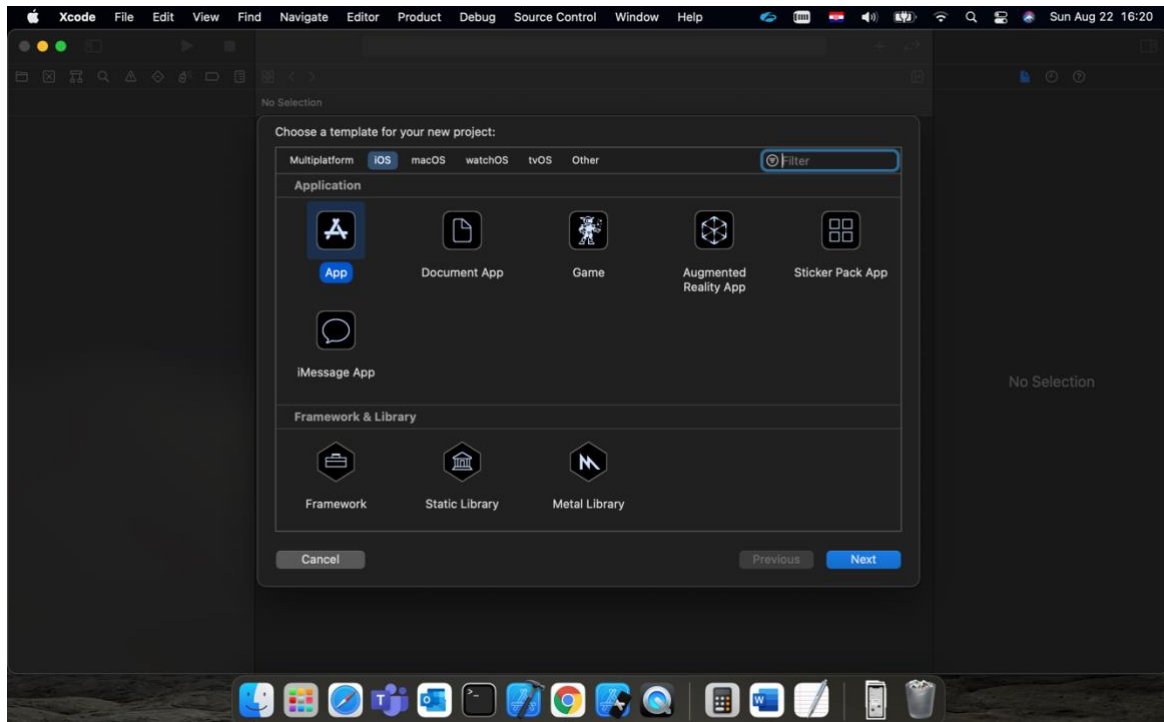
### 2.2.1. Kreiranje Xcode projekta

Nakon što smo preuzeli Xcode sa App Store-a potrebno je da kreiramo projekat. Projekat kreiramo na sljedeći način:

- Na slici 4. odaberemo opciju Create a new Xcode project

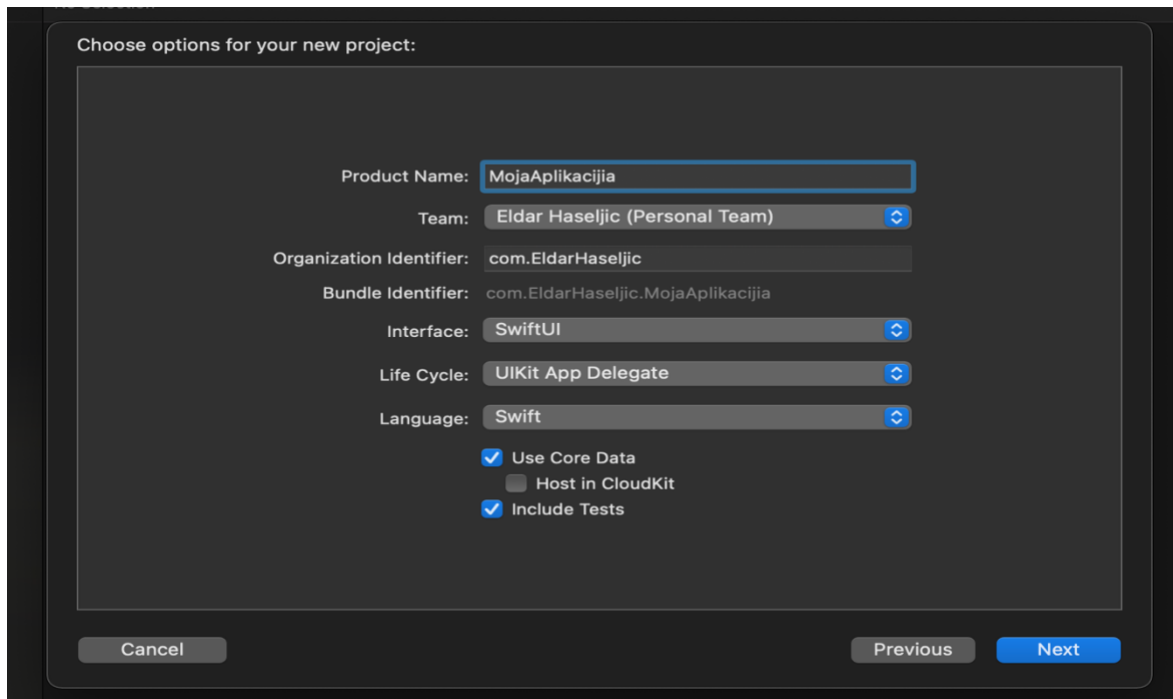


Slika 4. Welcome to Xcode



Slika 5. Xcode interface

- Na Slici 5.
  - Odaberemo ios za Multiplatform
  - Odaberemo App u Application sekciji
- Pritisnemo Next



Slika 6. Opcije novog Xcode projekta

- Na slici 6.
  - Unesemo Product Name
    - Naziv naše aplikacije
  - Odaberemo Team
    - Naši timovi za App Store Connect bi se trebali pojaviti ovdje ako smo povezali Xcode sa svojim Apple ID -om.
    - Ako imate besplatni račun programera, ovdje odaberite Lični tim. Što je učinjeno u slučaju naše aplikacije
  - Editujemo Organization Identifier (opcionalno)
    - Organization Identifier je obrnuti naziv domene koji jedinstveno identificira vašu aplikaciju (u App Storeu).
    - Uobičajeno je izabrati nešto poput naziva com.organization.name.app
  - Odaberemo Interface
    - Biramo UI framework koji želimo koristiti za izradu svoje aplikacije.
    - Možete birati između Storyboard (UIKit) ili SwiftUI.
    - Za našu aplikaciju je korišten Storyboard interface
  - Odaberemo Life Cycle
    - Ovo određuje pristup koji vaš Xcode projekt koristi za upravljanje životnim ciklusom vaše aplikacije, uključujući način na koji se početni User Interface vaše aplikacije pokreće.
    - Možete birati između UIKit App Delegate ili SwiftUI App.
    - Za našu aplikaciju je korišten UIKit App Delegate
  - Odaberemo Language
    - Ovdje se misli u kojem jeziku želimo da pišemo našu aplikaciju
    - Možemo birati između izrade aplikacije s Objective-C ili sa Swiftom.
    - Međutim za moju verziju XCoda je bilo ponuđeno samo Swift te smo i odabrali Swift opciju
  - Dodatne checkmark opcije
    - Ovo dodaje neki zadani kod vašoj aplikaciji.
    - Use Core Data služi za spremanje trajnih podataka vaše aplikacije za izvan mrežnu upotrebu te za keširanje privremenih podataka.
    - Include Tests se koristi u slučaju da želimo napisati određene test datoteke za testiranje naše aplikacije
    - Host in CloudKit se koristi ukoliko želimo da povežemo našu aplikaciju sa Apple CloudKit-om
    - Za našu aplikaciju smo odabrali samo Use Core Data opciju.
  - Pritisnemo Next i Create

Uspješno smo kreirali projekat i možemo početi sa radom

## 2.3. iOS Simulator

iOS simulator je alat koji nam služi kako bi pokrenuli i testirali aplikaciju na našem računalu. Simulator omogućuje simulaciju za većinu Apple uređaja te podržava opcije koje ti uređaji imaju. Simulator ima svoje prednosti i nedostatke. Koristan je za simuliranje i testiranje korisničkog okruženja koje se može obavljati interakcijom miša, međutim za simuliranje i testiranje složenijih programskih rješenja postaje nedovoljan. Cijela aplikacija ovog diplomskog rada je simulirana u iOS Simulatoru i simulator je mogao i uspio odraditi sve potrebne zahtjeve korisnika. Aplikacija je također testirana na fizičkom uređaju Iphone 8, OS verzija 13.7.



Slika 7. iOS Simulator

## 2.4. CocoaPods

CocoaPods predstavlja menadžera koji upravlja bibliotekama koje koristimo u našim Xcode projektima. Biblioteke koje koristimo u našim projektima navedene su u jednoj tekstualnoj datoteci pod nazivom Podfile. CocoaPods će riješiti zavisnosti između biblioteka, dohvatiti rezultirajući izvorni kod, a zatim ga povezati u Xcode radni prostor za izgradnju našeg projekta.

### 2.4.1. Povezivanje CocoaPods sa projektom

CocoaPods je izgrađen s Ruby-om i bit će instaliran sa zadanim Ruby-om dostupnim na macOS-u. Da bi smo instalirali CocoaPods potrebno je da pokrenemo sljedeći komadnu u terminal.

```
sudo gem install cocoapods
```

Kod 1. Komanda za instalaciju cocoaPods

Nakon što smo instalirali CocoaPods potrebno je da napravimo Podfile unutar našeg projekta je datoteka koja sadrži dodatne biblioteke i specifikacije vezane za projekta. Datoteka bi se jednostavno trebala nazvati Podfile.

To možemo izvršiti uz pomoć komande

```
pod init
```

### Kod 2. Komanda za inicijalizaciju Podfile datoteke

Prva linija koda u kreiranom fajlu potrebno je da bude “platform :ios, '12.0'” te ona predstavlja platformu i najnižu verziju koju podržavamo.

```
# Uncomment the next line to define a global platform for your project
platform :ios, '12.0'

target 'BosniaRoadTraffic' do
  # Comment the next line if you don't want to use dynamic frameworks
  use_frameworks!

  # Pods for BosniaRoadTraffic
  pod 'RxSwift'
  pod 'RxCocoa'
  pod 'FirebaseFirestoreSwift'
  pod 'Firebase'
  pod 'Firebase/Analytics'
end

deployment_target = '12.0'
post_install do |installer|
  installer.generated_projects.each do |project|
    project.targets.each do |target|
      target.build_configurations.each do |config|
        config.build_settings['IPHONEOS_DEPLOYMENT_TARGET'] =
deployment_target
      end
    end
    project.build_configurations.each do |bc|
      bc.build_settings['IPHONEOS_DEPLOYMENT_TARGET'] = deployment_target
    end
  end
end
```

### Kod 3. Sadržaj Podfile datoteke BosniaRoadTraffic aplikacije

Da bismo koristili CocoaPods, moramo definirati Xcode target s kojim ćemo ih povezati. Na primjer, u gore navedenom fajlu za našu iOS aplikaciju, to bi bio naziv naše aplikacije. U fajlu imamo target sekciju u kojoj piše „target 'BosniaRoadTraffic' do“ nakon te linije koda slijede takozvani CocoaPod-ovi koji predstavljaju dodatne biblioteke koje ćemo koristiti u našem projektu. One se navode u formatu „pod 'PODNAME'“. Nakon navedenih svih biblioteka slijedi end komanda na kraju.

Unutar Podfile datoteke možemo navesti i određene dodatne specifikacije kao što smo mi uradili gore u našem Podfile-u. Dodatni kod izvršava postavljanje „`deployment_target`” -a za sve biblioteke na 12.0. Nakon što smo kreirali sadržaj našeg Podfile datoteke potrebno je da izvršimo komadu

```
pod install  
pod update
```

**Kod 4. Komade za instaliranje i nadogradnju biblioteka koje su navedene u Podfile**

koje će instalirati sve navedene biblioteke u naš projekat i izvršiti nadogradnju na njihove najnovije verzije. Također te komade kreiraju MyApp.xcworkspace fajl koji koristimo dalje u razvoju naše aplikacije.

## 2.5. Reaktivno programiranje

Reaktivno programiranje je način programiranja u kojem jedna komponenta odašilje niz podataka dok se druga komponenta na njih pretplaćuje i sluša promjenu vrijednosti unutar toka, te ima odgovarajuću reakciju na njih. Reaktivno programiranje je uvedeno iz razloga što mobilni uređaji nisu dovoljno snažni za kompleksno računanje i teške poslove, pa dolazi do smrzavanja mobilnih aplikacija i slično. Protok podataka postoji sve dok se ne pošalje događaj greške ili događaj koji javlja da su poslani svi podaci.

### 2.5.1. RxSwift i RxCocoa Framework

RxSwift i RxCocoa dio su paketa jezičnih alata ReactiveX (Rx) koji obuhvaćaju više programskih jezika i platformi. RxSwift je Framework za interakciju sa programskim jezikom Swift, dok je RxCocoa framework koji čini API-je za Cocoa koji se koriste u iOS -u i OS X lakšim za upotrebu s reaktivnim tehnikama. Cocoa je razvojni Framework za izradu korisničkih okruženja za aplikacije i programe na operativnom sistemu OS X, dok je Cocoa Touch UI razvojni Framework za razvoj aplikacija na iOS-u (za iPhone, iPod Touch i iPad).

Neke od funkcionalnosti koje se koriste u aplikaciji su:

- **Observable** – pomoću ove funkcionalnosti odašilju se podaci, tj. stvaraju se asinhroni tokovi podataka.
- **Observer** – pomoću ove funkcionalnosti se pretplaćuje na asinhronu protoke podataka. Ukoliko se odašilje nekakva vrijednost, u pretplati se dobije njena vrijednost te se s njom može dalje manipulirati ili pozvati akciju kojoj je ta vrijednost potrebna.

- DisposeBag – Kada se deinit() pozove na objekt koji drži DisposeBag, svaki pretplatnik automatski se odjavljuje sa onoga na što je pretplaćen, bez ovoga moguće da dobijemo memory leak što vodi da bi aplikacija mogla pasti ili da aplikacija automatski padne.

RxSwift ima mnoštvo operatora pomoću kojih se može odraditi pretvorba podataka. Neki od njih su map, filter, flatMap i slično. Da bi smo dodali Rx biblioteke u naš projekat potrebno je u Podfile datoteku dodati sljedeće dvije linije koda

```
pod 'RxSwift'  
pod 'RxCocoa'
```

**Kod 5. Komande za importiranje Rx biblioteka unutar Podfile datoteke**

## 2.6. Firebase - Cloud Firestore baza podataka

Firebase je Google-ova platforma za razvoj web i mobilnih aplikacija. Platforma Firebase sastoji se od mnogo servisa čija je uporaba besplatna, a neki od njih su: Cloud Firestore, ML Kit, Cloud Functions, Authentication, Hosting, Cloud



# Cloud Firestore

**Slika 8. Cloud Firestore**

Storage, Realtime Database. U ovome radu korištena je samo Cloud Firestore funkcionalnost Firebase-a.

Cloud Firestore služi za pohranu i sinkronizaciju podataka između korisnika i uređaja na globalnoj razini koristeći NoSQL bazu podataka. Cloud Firestore pruža trenutnu sinkronizaciju podataka i izvanmrežnu podršku uz učinkovite upite prema bazi. Odlična integracija s drugim Firebase proizvodima omogućuje ubrzanu izgradnju aplikacija bez poslužitelja. Firestore u aplikaciji je korišten za spremanje pozicija radara, stanja na putevima, te ostalih informacija i detalja koji oni nose sa sobom.



Da bismo mogli koristiti Firebase, treba nam samo Google račun. Nakon što se prijavimo na Firebase s tim računom, potrebno je da kreiramo projekat na Firebase koji također moramo uvezati sa našim Xcode projektom.

Postupak kreiranja projekta na Firebase je sljedeći:

1. Odaberemo opciju “Add Project”
2. Unesemo naziv projekta te pritisnemo “Continue”
3. Uključiti Google Analytics (opcionalno)
4. Pritisnemo opciju “Create Project”

Nakon što je projekat kreiran potrebno je isti uvezati sa našim Xcode projektom. Postupak povezivanje Firebase projekta sa Xcode projektom je sljedeći:

1. Otvorimo Firebase project Overview
2. Odaberemo opciju iOS
3. Unesemo iOS bundle ID iz Xcode projekta, te pritisnemo “Register App”
4. Potrebno je da preuzmемо GoogleService-Info.plist
5. Dodamo GoogleService-Info.plist u naš Xcode projekat
6. Kreiramo Podfile (ukoliko nismo) te dodamo

```
pod 'FirebaseFirestoreSwift'
pod 'Firebase'
pod 'Firebase/Analytics'
```

**Kod 6. Komande za importovanje Firebase biblioteka unutar Podfile datoteke**

7. U terminal pozovemo `pod update`
8. Pozovemo `Firebase.configure()` unutar class AppDelegate u dole navedenoj funkciji i dodamo liniju `import Firebase` van klase

```
import Firebase

@main
class AppDelegate: UIResponder, UIApplicationDelegate {

    func application(_ application: UIApplication,
                    didFinishLaunchingWithOptions launchOptions:
[UIApplication.LaunchOptionsKey: Any]?) -> Bool {
        // Override point for customization after application launch.
        FirebaseApp.configure()
        return true
    }
}
```

**Kod 7. Konfiguracija Firebase baze unutar projekta**

9. Firebase uspješno podešen

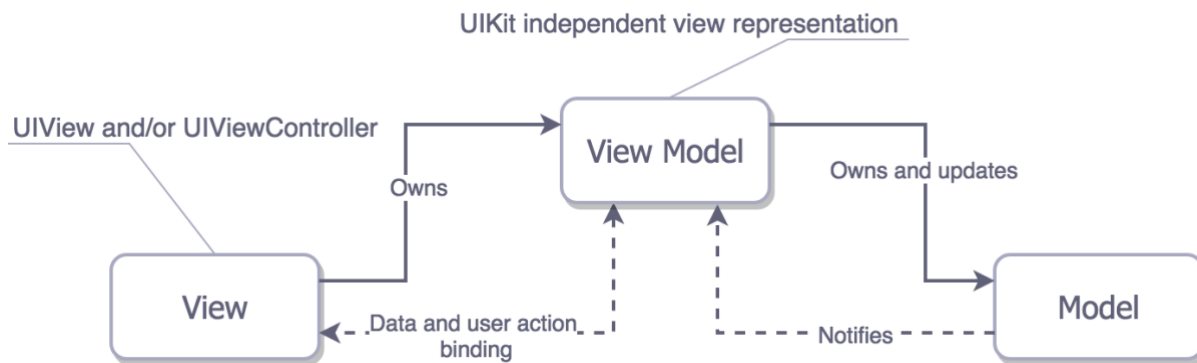
## 2.7. Model – View – ViewModel (MVVM)

Model-View-ViewModel je način dizajna programske arhitekture. Arhitektura programske podrške je plan koji opisuje skup aspekata i odluka koji su od iznimne važnosti za projekat. Podrazumijeva uzimanje u obzir sve vrste zahtjeva, organizaciju sistema, međusobnu komunikaciju između dijelova sistema i drugo.

Glavna karakteristika MVVM arhitekture je potpuna odvojenost poslovne logike od logike za postavljanje View-a. Unutar ViewModel-a se obavlja sva manipulacija nad podacima, tako da View ne zna ništa o podacima osim njihovog prikaza na korisničkom okruženju.

Kao što vidimo iz naziva, arhitektura se sastoji od tri dijela:

- ViewModel – sadrži poslovnu logiku aplikacije i sastoji se od različitih programskih dijelova ovisno o složenosti aplikacije.
- View – sadrži kod za prikaz vizualnih elemenata korisničkog okruženja i reakcije na korisničke akcije. Sastoji se od protokola ili evenata koji sadrže metode korisnikovih akcija i ViewController koji je odgovoran za materijalizaciju i prikazivanje komponenti korisničkog okruženja te otkrivanje događaja koje prosljeđuje viewModel-u.
- Model – sadrži podatke aplikacije. Najčešće se radi od klasama ili strukturama.

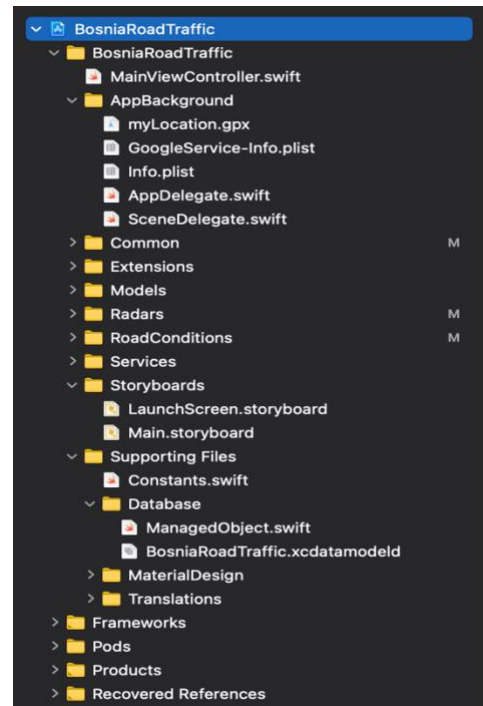


Slika 9. MVVM (Model – View- ViewModel) arhitektura

### 3. Struktura aplikacije

Sama aplikacija je pisana uz pomoć MVVM arhitekturom, a na slici 10. možemo vidjeti strukturu same aplikacije. Svaka funkcionalnost je odvojena u zasebnu datoteku te svaka funkcionalnost ima svoj View, Model i ViewModel.

Osim funkcionalnosti postoji i Manager koji nam služi za postavljanje komunikacije između uređaja i Firestore baze podataka koja se nalazi na Google servisima te veza i sa samim datotekama u kojim su implementirane klase i funkcionalnosti koje se koriste u cijeloj aplikaciji.



Slika 10. Struktura aplikacije

*AppDelegate.swift* je glavna tačka ulaska u aplikaciju. Metode AppDelegate pozivaju se za događaje u životnom ciklusu na razini aplikacije. U zadanom AppDelegate.swift postoje tri metode za koje Apple smatra da su važne koje moramo razmotriti i razmotrimo ih:

- **func application(\_: didFinishLaunchingWithOptions :) -> Bool**
  - Ova metoda se poziva kada se aplikacija pokrene i gdje se vrši postavljanje aplikacije. Tu smo izvršili konfiguraciju Firebase baze podataka.
- **func application(\_: configurationForConnecting: options :) -> UISceneConfiguration**
  - Ova metoda se poziva kada aplikaciji za prikaz bude potrebna nova scena ili prozor. Ova se metoda ne poziva pri pokretanju aplikacije, već samo kada je potrebno pribaviti novu scenu ili novi prozor.
- **func application (\_: didDiscardSceneSessions :)**
  - Ova metoda se poziva kad god korisnik odbaci scenu poput prevlačenja iz prozora za više zadataka ili ako se odbaci programski.
  - Ova se metoda poziva za svaku odbačenu scenu nedugo nakon što se pozove metoda (\_: didFinishLaunchingWithOptions :) ako se aplikacija ne izvodi kada korisnik odbaci scenu.

```

1: //
2: // AppDelegate.swift
3: // Bosnia Road Traffic
4: //
5: // Created by Eldar Haseljic on 1/3/21.
6: // Copyright © 2021 Eldar Haseljic. All rights reserved.
7: //
8:
9: import UIKit
10: import CoreData
11: import Firebase
12:
13: @main
14: class AppDelegate: UIResponder, UIApplicationDelegate {
15:
16:     func application(_ application: UIApplication,
17:                     didFinishLaunchingWithOptions launchOptions:
18: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {
19:         // Override point for customization after application launch.
20:         FirebaseApp.configure()
21:         return true
22:     }
23:
24:     // MARK: UISceneSession Lifecycle
25:
26:     @available(iOS 13.0, *)
27:     func application(_ application: UIApplication,
28:                     configurationForConnecting
29: connectingSceneSession: UISceneSession,
30:                     options: UIScene.ConnectionOptions) ->
31: UISceneConfiguration {
32:         // Called when a new scene session is being created.
33:         // Use this method to select a configuration to create the
34: new scene with.
35:         return UISceneConfiguration(name: "Default Configuration",
36: sessionRole: connectingSceneSession.role)
37:     }
38:
39:     @available(iOS 13.0, *)
40:     func application(_ application: UIApplication,
41:                     didDiscardSceneSessions sceneSessions:
42: Set<UISceneSession>) {
43:         // Called when the user discards a scene session.
44:         // If any sessions were discarded while the application was
45: not running, this will be called shortly after
46: application:didFinishLaunchingWithOptions.
47:         // Use this method to release any resources that were
48: specific to the discarded scenes, as they will not return.
49:     }
50: }

```

**Kod 8. Sadržaj AppDelegate.swift fajla**

Od verzije 13 pa nadalje, **SceneDelegate.swift** preuzima neke odgovornosti od AppDelegate.swift. Konkretno vezano za UIWindow iz AppDelegate.swift sada je UIScene u SceneDelegate. Aplikacija može imati više scena koje uglavnom obrađuju sučelje aplikacije i sadržaj aplikacije. Dakle, SceneDelegate.swift je odgovoran za ono što je prikazano na ekranu u terminima korisničkog sučelja i podataka. Unutar naše aplikacije nismo ništa mijenjali vezano za ovaj fajl, tako da je njegov sadržaj ostao isti onakav kakav bude kada se kreira sam porjekat. Samo ćemo nabrojati njegove zadane metode, te objasniti ih u par recenica, a te metode su:

- **scene(\_:willConnectTo:options:)**
  - Ovo je prva metoda koja se poziva u životnom ciklusu UISceneSession. Ova metoda će stvoriti novi UIWindow, postaviti glavni kontroler pogleda i učiniti ovaj prozor ključnim prozorom za prikaz.
- **sceneWillEnterForeground(\_:)**
  - Ova metoda se poziva kada se scena treba pokrenuti, primjerice kada aplikacija postane aktivna po prvi put ili pri prijelazu iz pozadine u prednji plan.
- **sceneDidBecomeActive(\_:)**
  - Ova metoda se poziva odmah nakon metode WillEnterForeground i ovdje je scena postavljena i vidljiva i spremna za upotrebu.
- **sceneWillResignActive(\_:)**
- **sceneDidEnterBackground(\_:)**
  - Ove se metode pozivaju pri stupnju aplikacije u pozadinu.
- **sceneDidDisconnect(\_:)**
  - Kad god se scena pošalje u pozadinu, iOS bi mogao odlučiti potpuno odbaciti scenu kako bi oslobodio resurse. To znači da je scena isključena iz sesije i nije aktivna. iOS može odlučiti ponovno povezati ovu scenu sa scenom kad korisnik tu scenu ponovno dovede u prvi plan. Ova se metoda može koristiti za odbacivanje svih resursa koji se više ne koriste.

Pored navedena dva glavna fajla za pokretanje aplikacije koji se nalaze unutar AppBackground datoteke kao što možemo vidjeti na slici 10. U strukturi su također važni :

- *BosniaRoadTraffic.xcdatamodeld*
  - *CoreData baza podataka na mobitelu*
- *GoogleService-Info.plist*
  - *Sadrži sve podatke potrebne Firebase iOS SDK -u za povezivanje s vašim Firebase projektom*
- *Info.plist*
  - *To je popis karakteristika čiji parovi ključ / vrijednost navode bitne informacije o konfiguraciji vremena izvođenja za aplikaciju.*

*Ostale fajlove i datoteke ćemo proći u narednim stranicama*

## 4. Korisničko okruženje aplikacije

Aplikacija razvijena u diplomskom radu podržava sve dimenzije iPhone uređaja. Također ima podršku korištenja aplikacije u vodoravnom (landscape) ili vertikalnom (portrait) načinu rada. Korisničko okruženje je razvijeno pisanjem programskog koda u Swiftu i uz pomoć Storyboard-ova. Storyboard je vizuelni prikaz korisničkog okruženja iOS aplikacije, koji prikazuje zaslone sadržaja i veze između tih ekrana.

Aplikacija ima nekoliko zaslona, a to su:

- Zaslون pokretanja
  - *LaunchScreen.storyboard*
- Glavni zaslon
  - *MainScreen.storyboard*
- Zaslon za prikaz radara
  - *RadarsMapStoryboard.storyboard*
- Zaslon za prikaz stanja na putevima
  - *RoadConditionsStoryboard.storyboard*
- Zaslon za prijavu novog radara/ stanja na putu
  - *ReportStoryboard.storyboard*
- Zaslon za filtriranje radara/stanja na putevima
  - *FilterStoryboard.storyboard*
- Zaslon za prikaz detalja vezanih za radar/stanje na putu i prikaz općih informacija za puteve
  - *DetailsStoryboard.storyboard*

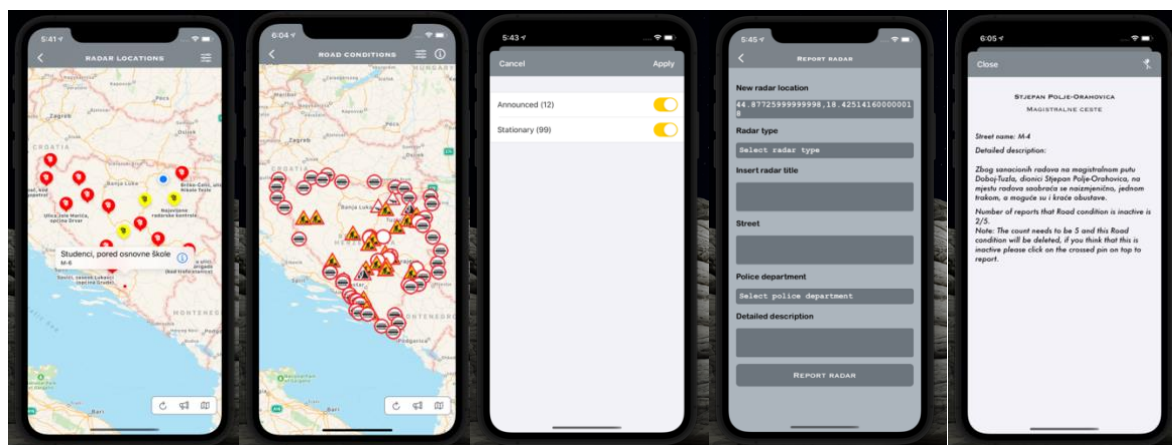
U nastavku rada ćemo razraditi samo detaljno sve funkcionalnosti svakog zaslona.



Slika 11. Zaslon pokretanja



Slika 12. Glavni zaslon



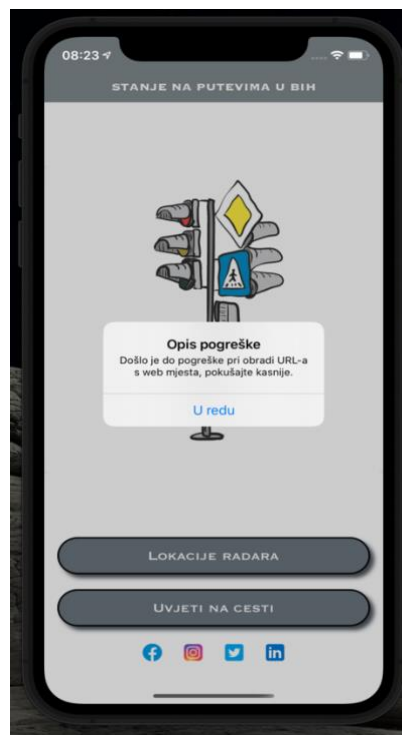
Slika 13. Prikaz ostalih screenova koji su vidljivi u aplikaciji

## 5. Opis rada i implementacija aplikacije

### 5.1. Opis rada glavnog zaslona

Kada korisnik pokrene aplikaciju na svom iPhone uređaju prva stvar na koju nailazi jeste zaslon pokretanja koji je prikazan na Slici 11. U suštini to je zaslon koji dobijemo kada kreiramo sam projekat. Razlika između zadane vrijednosti koju dobijemo prilikom kreiranja projekta i našeg zaslona pokretanja (*LaunchScreen.storyboard*) jeste što se na našem nalazi slika, dok na zadanoj vrijednosti imamo samo bijelu pozadinu ili crnu u zavisnosti da li je tamni način rada uključen. Nakon što se aplikacija učita dolazimo na glavni zaslon (*MainScreen.storyboard*) koji je na prikazan na Slici 12.

Na glavnom zaslonu možemo primijetiti na vrhu zaslona da ima naziv, dva siva dugmeta, te nekoliko sličica društvenih mreža koje također predstavljaju dugmad. Ukoliko pritisnemo na neku od sličica društvenih mreža, otvoriti će nam se zadani vanjski pretraživač na mobitelu (Safari inače) sa zadanom društvenom mrežom. Sve sličice društvenih mreža otvaraju društvenu mrežu osim LinkedIn sličice koja namjerno ima pogrešno podešen link, koji otvara upozorenje (Alert) gdje imamo informaciju da je link krivo podešen. To je urađeno samo za prikaz šta bi se desilo da je Link krivo podešen.



Slika 14. Prikaz upozorenja prilikom klika na LinkedIn

#### 5.1.1. Implementacija glavnog zaslona

Svaki klik na bilo koje navedeno dugme je izvršen uz pomoć tzv. Reaktivnog wrappera za TouchUpInside event nad dugmetom (rx.tap). Na njega predplaćujemo uz pomoć bind funkcije, gdje unutar nje pišemo šta želimo da se desi u slučaju klika na dugme. U slučaju da dođe do greške, bit će pokrenuta fatalError funkcija koja ruši aplikaciju, tako da je potrebno je koristiti sa oprezom. Svaki klik na većinu dugmadi u aplikaciji je izvršen na način koji je naveden u Kod 9. sekciji.

```

52: roadConditionsButton.rx.tap.bind { [unowned self] in
53:     pushViewController(viewController:
RoadConditionsViewController.showRoadConditions())
54:     }.disposed(by: disposeBag)
55:
56:     facebookButton.rx.tap.bind { [weak self] in
57:         guard let self = self else { return }
58:         self.openExternalUrl(urlString: Constants.URLPaths.facebookURL)
59:     }.disposed(by: disposeBag)

```

#### Kod 10. Reaktivni wrapper za TouchUpInside event nad dugmetom

Unutar prvog bind vidimo funkciju `pushViewController(viewController: RoadConditionsViewController.showRoadConditions())` koja služi za prelazak sa jednog zaslona na drugi pri čemu se oni pomijeraju u lijevu stranu tj vrši se push nad zaslonima kao da se nalaze u jednoj stack strukturi.

```

14: public func pushViewController(viewController: UIViewController) {
15:     navigationController?.pushViewController(viewController,
16:                                             animated: true)
17: }

```

#### Kod 9. Prikaz sadržaja funkcije pushViewController(viewController: \_)

Funkcija se nalazi unutar fajla `UIViewController+Extension.swift`, gdje su ujedno smještene sve tzv. extenzije za `UIViewController`. Taj fajl se nalazi u folderu na putanji `BosniaRoadTraffic/Extensions/UIExtensions` gdje se nalaze i razne druge extenzije za druge UI komponente, dok unutar folderu `BosniaRoadTraffic/Extensions` pored UI extenzija imamo i neke male extenzije za osnovne tipove kao sto su `Array`, `String`, `Date` i `Optional`. U dugom bind vidimo funkciju `openExternalUrl(urlString: _)` gdje nam sami naziv funkcije govori šta radi tj otvara odgovarajuću url u vanjskom pretraživaču. Također i ova funkcija se nalazi unutar fajla `UIViewController+Extension.swift`

```

67:     public func openExternalUrl(urlString: String) {
68:         guard let webURL = URL(string: urlString) else {
69:             self.presentAlert(title: ERROR_DESCRIPTION,
70:                             message: ERROR_URL_MESSAGE,
71:                             buttonTitle: OK,
72:                             handler: nil)
73:             return
74:         }
75:         UIApplication.shared.open(webURL)
76:     }

```

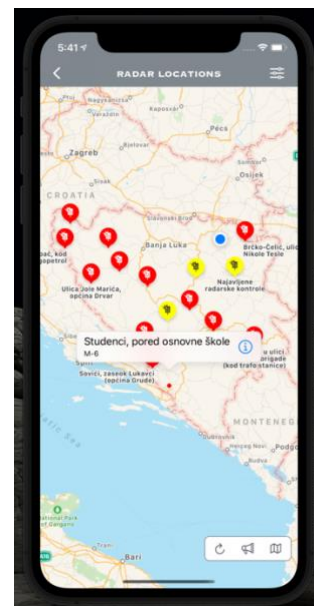
#### Kod 11. Sadržaj funkcije openExternalUrl(urlString: \_)

Funkcionalnost glavnog zaslona je definisana u fajlu `MainViewController.swift` i to je jedini zaslon koji nema `viewModel` jer mu nije bio potreban.



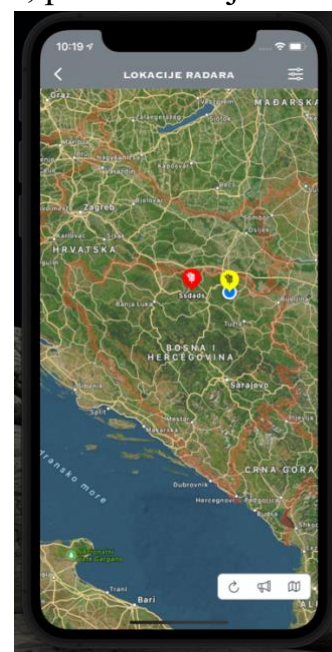
## 5.2. Opis rada zaslona za prikaz radara

Ukoliko pritisnemo na gornje sivo dugme na glavnom zaslonu, pokreće se spinner za učitavanje te prvo na što ćemo naići jeste Alert koji od nas traži da dopustimo prikaz naše lokacije na mapi, gdje korisnik može odabrati da uvijek ili samo jednom dopušta prikaz lokacije ili odbije prikaz lokacije. Ukoliko korisnik odbije prikaz lokacije biće vraćen automatski na glavni tj početni zaslon. Ukoliko je dopustio prikaz lokacije onda se poziva API poziv koji dohvata sve lokacije radara koje se nalaze trenutno u Firestore bazi, te se isti spremaju i u bazu na mobitel. Nakon što su radari učitani oni se prikazuju na mapi kao na slici 15. U gornjem desnom uglu korisnik ima opciju da filtriranje radara.



Slika 15. Zaslon sa prikazanim radarima

Radari se mogu filtrirati samo na dva načina tj stacionarni ili najavljeni radari. Ukoliko su vidljivi na mapi samo jedni ili nijedni onda je opcija za filtriranje isključena tj nije vidljiva. O filtriranju radara ćemo malo detaljnije u nastavku rada. Pored opcije za filtriranje u desnom donjem uglu imamo tri dodatne opcije tj dugmeta. Prvo dugme jeste za ponovno učitavanje zaslona, pri čemu osvježimo radare tj. učitamo nove ili se izbrišu određeni ako nisu više u Firebase bazi. Pored nje imamo opciju za prijavu tj ako smo uočili novi radar možemo ga prijaviti. O tome također detaljnije u nastavku. Zadnja opcija jeste promjena tipa mape tj imamo mapu kao što je prikazana na slici 15 i mapu sa terenom koja je prikazana na slici 16.



Slika 16. Zaslon sa prikazanim radarima, mapa teren

Ukoliko radara nemamo ili ako smo dobili informacije o novim radarima ili ako se desi neka pogreška uvijek ćemo dobiti odgovarajući Alert, tipa kao na slici 14.

Radari na mapi su označeni sa crvenom ili žutom bojom, crveni su stacionarni, dok su žuti najavljeni radari.

Ukoliko pritisnemo na neki od njih dobit ćemo kratke informacije o istom kao što je učinjeno na slici 15, dok ukoliko ponovo pritisnemo na taj mali skočni prozor otvorit će nam se zaslon sa malo detaljnijim informacijama vezanih za taj radar.

## 5.2.1. Implementacija zaslona za prikaz radara

Sa glavnog zaslona na zaslon sa radarima dolazimo nakon klika na gornje sivo dugme na glavnom zaslonu, tu se poziva funkcija `pushViewController(viewController: RadarsMapViewViewController.showRadars())`.

`pushViewController(viewController:_)` funkciju smo objasnili ranije, te znamo da služi sa prikaz novg zaslona sada je potrebno da pogledamo sadržaj funkcije `showRadars()`.

```
233: extension RadarsMapViewViewController {
234: static func getViewController() -> RadarsMapViewViewController {
235: return
  UIStoryboard(name:Constants.StoryboardIdentifiers.RadarsMapStoryboard,
236:               bundle: nil)
237: .instantiateViewControllerWithIdentifier(RadarsMapViewViewController.self)
  !
238:   }
239:
240: static func showRadars() -> RadarsMapViewViewController {
241: let radarsViewController = RadarsMapViewViewController.getViewController()
242:   radarsViewController.setViewModel()
243:   return radarsViewController
```

**Kod 12. Prikaz funkcije `showRadars()`**

U `showRadars()` funkciji možemo primijetiti da se prvo dohvata `viewController` koji je `RadarsMapViewViewController`. Nakon toga postavljamo `viewModel` za taj kontroler i vratimo kontroler nazad. U to vrijeme unutar postavljenog `viewModela`, koji u ovom slučaju predstavlja `RadarsMapViewModel`, se obavlja funkcionalnost koja na zaslon za korisnika izbacuje poruku da korisnik dopusti aplikaciji pristup njegovoj lokaciji. To se izvršava unutar funkcije `checkLocationServices()`.

```
57: func checkLocationServices() {
58:   if CLLocationManager.locationServicesEnabled() {
59:     setupLocationManager()
60:     checkLocationAuthorization()
61:   } else {
62: messageTransmitter.onNext(Adviser(title: ERROR_DESCRIPTION,
63:                                   message: LOCATION_SERVICE_DISABLED,
64:                                   isError: true))
65:   }
66: }
67:
68: private func setupLocationManager() {
69:   locationManager.delegate = self
70:   locationManager.desiredAccuracy = kCLLocationAccuracyBest
71: }
```

**Kod 13. Sadržaj `checkLocationServices()` funkcije**

Prvo što primijetimo unutar ove funkcije jeste da provjeravamo da li je uopšte uključena opcija lokacije globalno na mobitelu ukoliko nije korisnik će dobiti poruku da bi trebao uključiti Lokaciju na svom mobitelu. Poruku će dobiti preko `messageTransmitter` koji je `PublishSubject<Adviser>()` objekat i on se može koristiti i kao odašiljač ili prijemnik informacija između `ViewCotrollera` i `viewModela`. Pošto je `PublishSubject` generičkog tipa vidimo da u našem slučaju uzima objekat `Adviser`. Model te strukture je prikazan u Kod 14.

```
9: import Foundation
10: import UIKit
11:
12: class Adviser {
13:     var title: String
14:     var message: String
15:     var isError: Bool
16:
17:     init(title: String,
           message: String,
           isError: Bool = false) {
18:         self.title = title.localizedUppercase
19:         self.message = message
20:         self.isError = isError
21:     }
```

#### Kod 14. Sadržaj klase Adviser

Unutar navedene strukture imamo tri parametra to su:

- title
  - predstavlja naziv našeg Alerta kojeg prikazujemo useru
- message
  - predstavlja poruku koju želimo proslijediti našem korisniku
- isError
  - predstavlja parametar koji je inače false jer smatramo da je Alert samo poruka za korisnika, međutim ovu strukturu koristimo I kada želimo da prikazemo korisniku da se desila određena pogreška, tako da se taj parametar koristi u takvim situacijama

Ukoliko lokacija kod korisnika na mobitelu je uključena izvrše se određene postavke za `LocationManagera`, te se pozove funkcija `checkLocationAuthorization()` koja provjerava da li je korisnik odbio prikazati lokaciju ili je označio da je lokacija vidljiva samo jednom ili neke druge opcije lokacije. U zavisnosti od opcije šaljemo informaciju preko `onNext(_ element: Element)` metoda, koja se poziva nad variablom `userLocationStatus` koja je tipa `PublishSubject<AuthorizationStatus>()` `viewControlleru`. `ViewController` je već ranije unutar `setupObservers()` funkcije pozvao `bind` metod nad objektom iz `viewModela`.

Sadržaj `checkLocationAuthorization()` funkcije možemo vidjeti u Kod 15 sekciji.

```
73: private func checkLocationAuthorization() {
74:     switch CLLocationManager.authorizationStatus() {
75:     case .authorizedWhenInUse:
76:         userLocationStatus.onNext(.authorizedWhenInUse)
77:         currentAuthorizationStatus = .authorizedWhenInUse
78:     case .denied:
79:         userLocationStatus.onNext(.denied)
80:         currentAuthorizationStatus = .denied
81:     case .notDetermined:
82:         locationManager.requestWhenInUseAuthorization()
83:         currentAuthorizationStatus = .notDetermined
84:     case .restricted:
85:         userLocationStatus.onNext(.restricted)
86:         currentAuthorizationStatus = .restricted
87:     case .authorizedAlways:
88:         userLocationStatus.onNext(.authorizedAlways)
89:         currentAuthorizationStatus = .authorizedAlways
90:     @unknown
91:     default:
92:         userLocationStatus.onNext(.error)
93:     }
94: }
```

**Kod 15. Sadržaj funkcije `checkLocationAuthorization()`**

```
98: viewModel.userLocationStatus.bind(onNext: { [unowned self] isVisible
in
99:     switch isVisible {
100:     case .authorizedWhenInUse,
101:     .authorizedAlways:
102:         mapView.showsUserLocation = true
103:         viewModel.fetchData()
104:     case .denied:
105:         presentAlert(title: ERROR_DESCRIPTION,
106:             message: String(format:
LOCATION_SERVICE, AuthorizationStatus.denied.translation),
107:             buttonTitle: OK,
108:             handler: { _ in tapBackButton(self) })
109:     case .restricted:
110:         presentAlert(title: ERROR_DESCRIPTION,
111:             message: String(format:
LOCATION_SERVICE, AuthorizationStatus.restricted.translation),
112:             buttonTitle: OK,
113:             handler: { _ in tapBackButton(self) })
114:     case .error:
115:         presentAlert(title: ERROR_DESCRIPTION,
116:             message: String(format:
LOCATION_SERVICE, UNKNOWN),
117:             buttonTitle: OK,
118:             handler: { _ in tapBackButton(self) })
119:     case .notDetermined: break
120:     }
121: })
122: .disposed(by: disposeBag)
```

**Kod 16. Kod za slušanje promjene vrijednosti variable `userLocationStatus`**

Ako je lokacija odobrena jednom ili uvijek za aplikaciju u Kod 16. Sekciji vidimo da se poziva funkcija `fetchData()`, koja unutar sebe koristi `managera` koji je u ovom slučaju `MainManager` koji poziva funkciju `getRadars()` koja dalje dohvata podatke o radarima iz `Firestore`.

```
123: func fetchData() {
124:     manager.getRadars { [weak self] (response,errorAdviser) in
125:         guard
126:             let self = self,
127:             let response = response,
128:             let errorAdviser = errorAdviser
129:         else {
130:             print("Radars updated successfully")
131:             return
132:         }
133:
134:     DispatchQueue.main.async {
135:         self.showRadars(radars: response, errorAdviser: errorAdviser)
136:     }
137: }
138: }
```

**Kod 17. Sadržaj funkcije `fetchData()`**

Ukoliko korisnik dohvata prvi put podatke iz baze, pozove se api poziv za dohvaćanje podataka iz `Firestore`, pri čemu se oni dekodiraju u `Radar` strukturu te se spremaju u bazu u `mobitel`. Ukoliko se neki od radara nalazi u bazi na `mobitelu` a nismo ga ponovo dobili u `response` od `firestore` onda se takvi radari brišu iz baze u `mobitelu`. Također ukoliko u `response` dobijemo radar koji nema koordinate lokacije ili ima 5 prijava da je neaktivan i taj se ne unosi u bazu u `mobitel`, te se šalje i zahtjev za brisanje istog iz baze na `Firestore`. Ukoliko korisnik nije konektovan na internet biće mu dohvaćeni radari koje ima trenutno u bazi na `mobitelu` uz odgovarajuću poruku. U slučaju bilo kakve greške korisniku se šalje poruka greške. Kod za dohvaćanje podatka iz `Firestore` je prikazan u sekciji Kod 21. Nakon što su radari spremljeni u bazu šalju se na prikaz na mapu u funkciji `showRadars(radars: [Radar], errorAdviser: Adviser? = nil)`.

Za prikaz određenog objekta na mapi potrebno je da ta struktura nasljeđuje `MKAnnotation` interface, kao što je vidljivo za `Radar` objekat u kod 19. sekciji. `MKAnnotation` interface sadrži samo tri variable koje su vidljive u kod 18. Sekciji.

```
9 public protocol MKAnnotation : NSObjectProtocol {
10
11
12     // Center latitude and longitude of the annotation view.
13     // The implementation of this property must be KVO compliant.
14     var coordinate: CLLocationCoordinate2D { get }
15
16
17     // Title and subtitle for use by selection UI.
18     optional var title: String? { get }
19
20     optional var subtitle: String? { get }
21 }
22
```

**Kod 18. Sadržaj `MKAnnotation` interface-a**

```

29: public class Radar: NSObject, MKAnnotation {
30:
31:     @NSManaged public var policeDepartmentID: NSNumber?
32:     @NSManaged public var policeDepartmentName: String?
33:     @NSManaged public var coordinates: String?
34:     @NSManaged public var id: String?
35:     @NSManaged public var road: String?
36:     @NSManaged public var text: String?
37:     @NSManaged public var title: String?
38:     @NSManaged public var type: NSNumber?
39:     @NSManaged public var validFrom: String?
40:     @NSManaged public var validTo: String?
41:     @NSManaged public var numberOfDeletions: NSNumber
42:
43:     public var locationName: String?
44:     public var subtitle: String?
45:     var markerTintColor: UIColor
46:     var image: UIImage
47:     var radarType: RadarType
48:     public var coordinate: CLLocationCoordinate2D
49:     public var shouldDeleteRadar: Bool
50:     public override var description: String
51: }

```

### Kod 19. Sadržaj Radars strukture

```

12: class RadarMarkerView: MKMarkerAnnotationView {
13:
14:     override var annotation: MKAnnotation? {
15:         didSet {
16:             guard let radar = newValue as? Radar else { return }
17:             canShowCallout = true
18:             calloutOffset = CGPoint(x: 0, y: 0)
19:             markerTintColor = radar.markerTintColor
20:             detailCalloutAccessoryView =
getDescriptionLabel(text: radar.subtitle)
21:             rightCalloutAccessoryView = UIButton(type:
.detailDisclosure)
22:             glyphImage = radar.image
23:         }
24:     }
25:
26:     func getDescriptionLabel(text: String?) -> UILabel {
27:         let detailLabel = UILabel()
28:         detailLabel.numberOfLines = 1
29:         detailLabel.font = detailLabel.font.withSize(12)
30:         detailLabel.text = text
31:         return detailLabel
32:     }
33: }

```

### Kod 20. RadarMarkerView objekat za konfiguraciju radar oznake na mapi

```

140: func getRadars(_ completion: ((_ success: [Radar]?, _ errorAdviser: Adviser?) ->
Void)?) {
141:     let errorAdviser: Adviser = Adviser(title: ERROR_DESCRIPTION, message:
String())
142:     let connectionStatus = Reachability.isConnectedToNetwork()
143:     let writeManagedObjectContext = persistanceService.backgroundContext
144:     let oldRadars = fetchRadars(objectContext: writeManagedObjectContext)
145:     print("Number of old radars: \(oldRadars.count) \n \(oldRadars)")
146:     switch connectionStatus {
147:     case true:
148:         firestoreDataBase.collection("Radars").getDocuments() { (querySnapshot,
err) in
149:             if let err = err {
150:                 errorAdviser.message = err.localizedDescription
151:                 completion?([], errorAdviser)
152:                 return
153:             } else {
154:
155:                 writeManagedObjectContext.perform {
156:
157:                     guard let radars = querySnapshot?.documents else {
158:                         errorAdviser.message =
CustomError.canNotProcessData.errorDescription
159:                         completion?(oldRadars, errorAdviser)
160:                         return
161:                     }
162:
163:                     var newRadarsIDs: [String] = []
164:                     for currentRadar in radars {
165:                         guard let radarID = currentRadar[RadarJSON.id.rawValue] as?
String else { continue }
166:                         newRadarsIDs.append(radarID)
167:                         let newRadar = Radar.findOrCreate(radarID, context:
writeManagedObjectContext)
168:                         newRadar.fillRadarInfo(currentRadar.data())
169:                         if newRadar.shouldDeleteRadar {
170:                             writeManagedObjectContext.delete(newRadar)
171:                         }
172:                     }
173:
174:                     for oldRadar in oldRadars {
175:                         if let oldRadarID = oldRadar.id,
176:                            !newRadarsIDs.contains(oldRadarID) {
177:                             writeManagedObjectContext.delete(oldRadar)
178:                             self.deleteElement(elementId: oldRadarID, type: .radar)
179:                         }
180:                     }
181:
182:                     let status = writeManagedObjectContext.saveOrRollback()
183:                     if status {
184:                         if radars.isEmpty {
185:                             errorAdviser.title = RADARS_INFO
186:                             errorAdviser.message = NO_RADARS_FOUND
187:                             completion?(oldRadars, errorAdviser)
188:                         }
189:                         completion?(nil, nil)
190:                     } else {
191:                         errorAdviser.message =
CustomError.dataBaseError.errorDescription
192:                         completion?(oldRadars, errorAdviser)
193:                     }
194:                 }
195:             }
196:         }
197:     case false:
198:         errorAdviser.title = RADARS_INFO
199:         errorAdviser.message = YOU_ARE_CURRENTLY_OFFLINE
200:         completion?(oldRadars, errorAdviser)
201:     }
202: }

```

**Kod 21. Funkcija za dohvaćanje radara sa apia**

```

115: private func showRadars(radars: [Radar], errorAdviser: Adviser? = nil) {
116:     radarsInDatabase = radars
117:     radarsArray.onNext(radarsInDatabase)
118:     if let errorAdviser = errorAdviser {
119:         messageTransmitter.onNext(errorAdviser)
120:     }
121: }

```

### Kod 23. showRadars funkcija za prikaz radara na mapi

showRadars(radars: [Radar], errorAdviser: Adviser? = nil) funkcija vidimo da nad variablom radarsArray, koja je tipa PublishSubject<[Radar]>(), poziva metod onNext(\_ element: Element), te se preko njega šalju radari viewControlleru za prikaz. Ovdje vidimo i da ako smo dobili neku grešku unutar funkcije za dohvaćanje i nju šalјemo viewControlleru, a tamo vidimo da se radari šalјu u prepareMapAndFilter funkciju u Kod 23. sekciji.

```

124: viewModel.radarsArray.bind(onNext: { [unowned self] radars in
125:     print("Number of new radars: \(radars.count) \n \(radars)")
126:     prepareMapAndFilter(with: radars)
127: })
128: .disposed(by: disposeBag)

```

### Kod 22. Predplatnik unutar viewControllera koji ceka na izmjene radarsArray variable

```

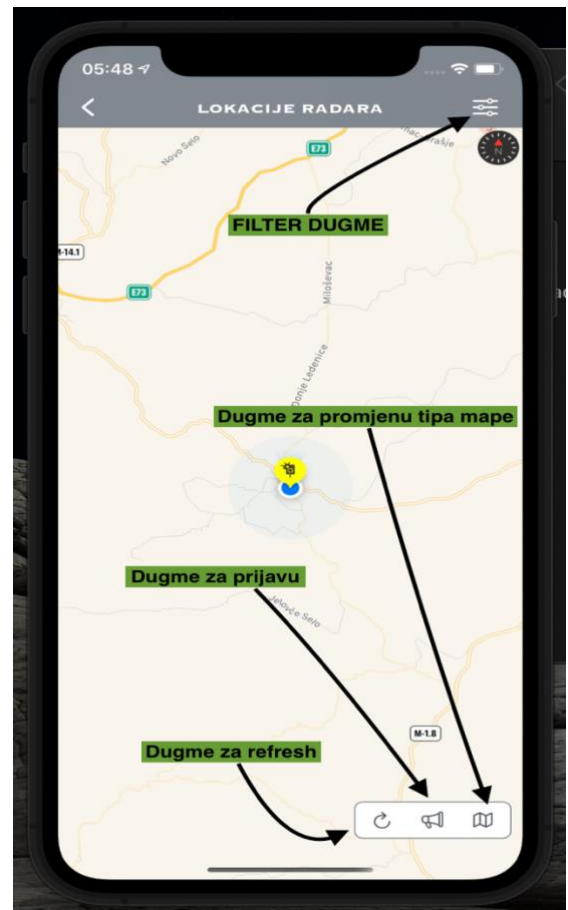
178: private func prepareMapAndFilter(with radars: [Radar]) {
179:     mapView.removeAnnotations(mapView.annotations)
180:     mapView.addAnnotations(radars)
181:
182:     if let currentLocation = viewModel.userCurrentLocation {
183:         mapView.setRegion(currentLocation, animated: true)
184:     }
185:
186:     let filterViewModel = FilterViewModel(radars: radars)
187:     if filterViewModel.numberOfFilters > 1 {
188:         filterViewController.setData(viewModel: filterViewModel,
189:                                     filterType: .radars)
190:         navigationItem.rightBarButtonItem = filterButton
191:         navigationItem.rightBarButtonItem?.isEnabled = true
192:     }
193:
194:     reportButton.isHidden = !Reachability.isConnectedToNetwork()
195:
196:     loadingIndicatorView.stopAnimating()
197:     delay(Constants.Time.TenSeconds) { [weak self] in
198:         guard let self = self else { return }
199:         self.reloadMapButton.isEnabled = true
200:     }

```

### Kod 24. Sadržaj prepareMapAndFilter funkcije unutar RadarsViewControllera



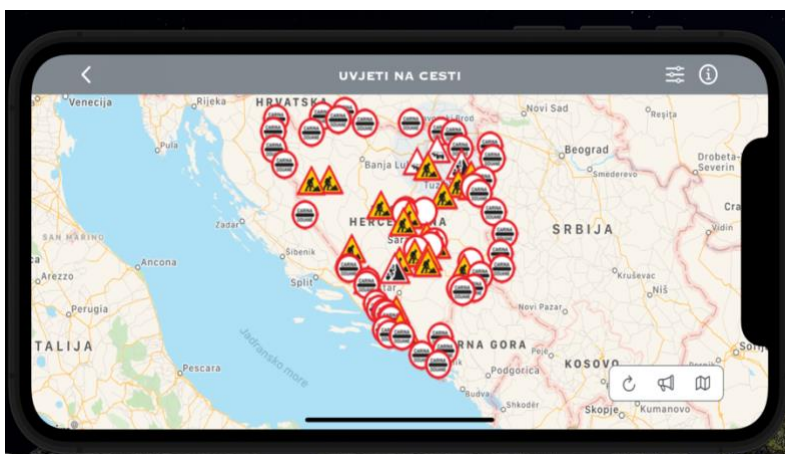
`prepareMapAndFilter(with radars: [Radar])` funkcija prvo obriše prethodne radara koji su prikazani na mapi, ovo je dodano iz razloga što korisnik na mapi ima dugme za refresovanje screena i screen se refresuje ukoliko pošaljemo aplikaciju u pozadinu. Nakon toga se dodaju svi radari na mapu, postavlja se lokacija samog korsika tj postavlja se region vidljivosti na krug od 20000. Pored toga postavljamo i filtere, tj ako korisnik ima različite vrste radara unutar baze, onda je dugme za filtriranje i omogućeno u suprotnom ta opcija neće biti vidljiva. Također u zavisnosti ako je korisnik konektovan na internet imati će mogućnost da prijavljuje nove radare tako da će i report dugme biti uključeno. Kada se sve poziva se delay koji radi u pozadini te nakon 10 sekundi, korisniku je omogućeno i korištenje refresh dugmeta, jer nema smisla da refresh se izvršava ako su podaci tek učitani. Kao što sam ranije naveo imamo i dugme za promjenu tipa mape tj da li zelimo da gledamo teren ili samo puteve.



Slika 17. Pozicije osnovnih dugmadi na mapi

### 5.3. Opis rada zaslona za prikaz stanja na putevima

Zaslon za prikaz stanja na putevima je po funkcionalnosti sličan zaslonu za prikaz radara. Opcije koje su vidljive na tom zaslonu su kao i na zaslonu za radare a to



Slika 18. Prikaz zaslona za prikaz stanja na putevima

su opcije za refresh, prijavu novog stanja na putu, filtriranje stanja na putu, tip mape te klik na bilo koje stanje na put će nam otvoriti prvo kratki opis pa tek onda detaljniji opis. Razlike jesu u prikazu stanja na putevima, te info opcija koju vidimo na slici 18 u desnom ćošku pored filter opcije.

### 5.3.1. Implementacija zaslona za prikaz stanja na putevima

Sa glavnog zaslona na zaslon sa radarima dolazimo nakon klika na donje sivo dugme na glavnom zaslonu, tu se poziva funkcija `pushViewController(viewController: RoadConditionsViewController.showRoadConditions()).pushViewController(viewController:_)` funkciju smo objasnili ranije, te znamo da služi sa prikaz novog zaslona sada je potrebno da pogledamo sadržaj funkcije `showRoadConditions()`.

```
252: extension RoadConditionsViewController {
253:     static func getViewController() -> RoadConditionsViewController {
254:         return UIStoryboard(name:
Constants.StoryboardIdentifiers.RoadConditionsStoryboard, bundle: nil)
255: .instantiateViewControllerWithIdentifier(RoadConditionsViewController.
self)!
256:     }
257:
258:     static func showRoadConditions() -> RoadConditionsViewController
{
259:         let roadConditionsViewController =
RoadConditionsViewController.getViewController()
260:         roadConditionsViewController.setViewModel()
261:         return roadConditionsViewController
262:     }
263: }
```

**Kod 25. Sadržaj funkcije showRoadConditions()**

Kao što možemo primijetiti u Kod 25. sekciji otvaranje zaslona za prikaz stanja na putevima ima skoro pa isti sadržaj kao i sekcija Kod 12. Razlika jeste u samom nazivu viewCotrollera jer je ovdje `RoadConditionsViewController` i `viewModel` jeste `RoadConditionsViewModel`. Implementacija se dalje nastavlja kao i za zaslon sa radarima, tj prvo provjeravamo lokaciju što je objašnjeno u sekcijama Kod 13, 15 i 16. Ukoliko je lokacija dopuštena prelazimo na dohvaćanje lokacija gdje se nalaze neki radovi ili druga stanja na putevima, tj poziva se funkcija `fetchData` iz `RoadConditionsViewModel`-a. Prva funkcija koja se poziva unutar te funkcije jeste api poziv za dohvaćanje opštih informacija o stanjima na putevima tj funkcija `getRoadConditionFullReport()`, te nakon što se ona završi poziva se funkcija `getRoadConditions()` koja dohvata sve lokacije, i vraća ih dalje `viewModelu` na obradu te se postupak nastavlja kao i kod prikaza radara. Razlika jeste što se response funkcije `getRoadConditionFullReport()` sprema u objekat tipa `RoadConditionDetails` a response od funkcije `getRoadConditions()` se sprema u objekat tipa `RoadCondition`. Svi objekti se i ovaj put spremaju i u bazu od mobitela.

```

154: func fetchData() {
155:     manager.getRoadConditionFullReport() { [weak self] in
156: self?.manager.getRoadConditions { [weak self] (response,
errorAdviser) in
157:         guard
158:             let self = self,
159:             let response = response,
160:             let errorAdviser = errorAdviser
161:         else {
162:             print("Road Conditons updated successfully")
163:             return
164:         }
165:
166:         DispatchQueue.main.async {
167:             self.showRoadConditons(roadConditions: response,
168:                                     errorAdviser: errorAdviser)
169:         }
170:     }
171: }
172: }

```

**Kod 26. Sadržaj funkcije fetchData() unutar RoadConditionsViewModel**

```

24: public class RoadConditionDetails: NSManagedObject {
25:
26:     @NSManaged public var id: String?
27:     @NSManaged public var title: String?
28:     @NSManaged public var validFrom: String?
29:     @NSManaged public var validTo: String?
30:     @NSManaged public var text: String?
31:     @NSManaged public var roadConditionID: NSNumber?
32:     @NSManaged public var roadConditionType: String?
33:     @NSManaged public var numberOfDeletions: NSNumber
34:
35:     public override var description: String
36: }

```

**Kod 27. Sadržaj RoadConditionDetails objekta**

```

172: private func reloadScreen() {
173:     setReportPinVisibility()
174:     loadingIndicatorView.startAnimating()
175:     reloadMapButton.isEnabled = false
176:     navigationItem.rightBarButtonItemItems?.forEach {
177:         $0.isEnabled = false
178:     }
179:     viewModel.fetchData()
180: }

```

**Kod 28. Sadržaj reloadScreen funkcije**

```

268: func getRoadConditionFullReport(_ completion: (() -> Void)?) {
269: let writeManagedObjectContext = persistenceService.backgroundContext
270: let connectionStatus = Reachability.isConnectedToNetwork()
271: let oldRoadConditions = fetchRoadFullReport(objectContext:
writeManagedObjectContext)
272:     switch connectionStatus {
273:     case true:
274: firestoreDataBase.collection("RoadConditionReport").getDocuments() {
(querySnapshot, _) in
275:         guard let roadConditionFullReports =
querySnapshot?.documents else {
276:             completion?()
277:             return
278:         }
279:
280:         writeManagedObjectContext.perform {
281:             if roadConditionFullReports.isEmpty {
282:                 oldRoadConditions.forEach({
283:                     writeManagedObjectContext.delete($0)
284:                 })
285:             } else {
286:                 for roadConditionFullReport in
roadConditionFullReports {
287:                     guard let roadConditionFullReportID =
roadConditionFullReport[RoadConditionDetailsJSON.id.rawValue] as? String
else { continue }
288:                     let newReport =
RoadConditionDetails.findOrCreate(roadConditionFullReportID,
289: context: writeManagedObjectContext)
290: newReport.fillInfo(roadConditionFullReport.data())
291:                 }
292:             }
293:
294:             let _ =
writeManagedObjectContext.saveOrRollback()
295:             completion?()
296:         }
297:     }
298:     case false:
299:         completion?()
300:         return
301:     }
302: }

```

**Kod 29. Sadržaj funkcije getRoadConditionFullReport**

```

204: func getRoadConditions(_ completion: ((_ success: [RoadCondition]?, _ errorAdviser: Adviser?) -> Void)?) {
205:     let errorAdviser: Adviser = Adviser(title: ERROR_DESCRIPTION, message: String())
206:     let connectionStatus = Reachability.isConnectedToNetwork()
207:     let writeManagedObjectContext = persistenceService.backgroundContext
208:     let oldRoadConditions = fetchRoadConditions(objectContext: writeManagedObjectContext)
209:     print("Number of old radars: \(oldRoadConditions.count) \n \(oldRoadConditions)")
210:     switch connectionStatus {
211:     case true:
212:         firestoreDataBase.collection("RoadConditions").getDocuments() { (querySnapshot, err) in
213:             if let err = err {
214:                 errorAdviser.message = err.localizedDescription
215:                 completion?([], errorAdviser)
216:                 return
217:             } else {
218:
219:                 writeManagedObjectContext.perform {
220:
221:                     guard let roadConditions = querySnapshot?.documents else {
222:                         errorAdviser.message = CustomError.canNotProcessData.errorDescription
223:                         completion?(oldRoadConditions, errorAdviser)
224:                         return
225:                     }
226:
227:                     var newRoadConditionsIDs: [String] = []
228:                     for currentCondition in roadConditions {
229:                         guard let conditionID = currentCondition[RoadConditionJSON.id.rawValue] as? String else { continue }
230:                         newRoadConditionsIDs.append(conditionID)
231:                         let newCondition = RoadCondition.findOrCreate(conditionID, context: writeManagedObjectContext)
232:                         newCondition.fillConditionInfo(currentCondition.data())
233:                         if newCondition.shouldDeleteRoadCondition {
234:                             writeManagedObjectContext.delete(newCondition)
235:                         }
236:                     }
237:
238:                     for oldCondition in oldRoadConditions {
239:                         if let oldConditionID = oldCondition.id,
240:                            !newRoadConditionsIDs.contains(oldConditionID) {
241:                             writeManagedObjectContext.delete(oldCondition)
242:                             self.deleteElement(elementId: oldConditionID, type: .roadCondition)
243:                         }
244:                     }
245:
246:                     let status = writeManagedObjectContext.saveOrRollback()
247:                     if status {
248:                         if roadConditions.isEmpty {
249:                             errorAdviser.title = ROAD_CONDITIONS_INFO
250:                             errorAdviser.message = NO_ROAD_CONDITIONS_FOUND
251:                             completion?(oldRoadConditions, errorAdviser)
252:                         }
253:                         completion?(nil, nil)
254:                     } else {
255:                         errorAdviser.message = CustomError.dataBaseError.errorDescription
256:                         completion?(oldRoadConditions, errorAdviser)
257:                     }
258:                 }
259:             }
260:         }
261:     case false:
262:         errorAdviser.title = ROAD_CONDITIONS_INFO
263:         errorAdviser.message = YOU_ARE_CURRENTLY_OFFLINE
264:         completion?(oldRoadConditions, errorAdviser)
265:     }
266: }

```

**Kod 30. Sadržaj funkcije getRoadConditions()**

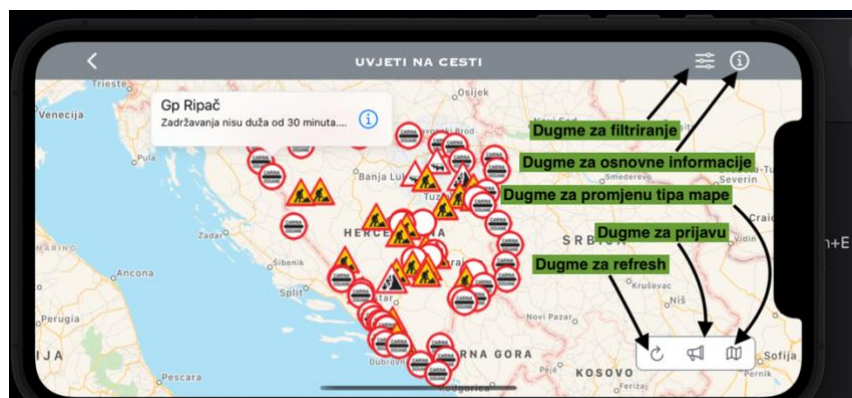
```

29: public class RoadCondition: NSObject, MKAnnotation {
30:
31:     @NSManaged public var roadID: NSNumber?
32:     @NSManaged public var roadType: String?
33:     @NSManaged public var coordinates: String?
34:     @NSManaged public var iconTitle: String?
35:     @NSManaged public var id: String?
36:     @NSManaged public var road: String?
37:     @NSManaged public var text: String?
38:     @NSManaged public var title: String?
39:     @NSManaged public var validFrom: String?
40:     @NSManaged public var validTo: String?
41:     @NSManaged public var numberOfDeletions: NSNumber
42:
43:     public var locationName: String?
44:     public var subtitle: String?
45:     var isCoordinateZero: Bool
46:     var hasNoIcon: Bool
47:     var shouldDeleteRoadCondition: Bool
48:     public var coordinate: CLLocationCoordinate2D
49:     var image: UIImage
50:     var roadConditionType: ConditionType
51:     public override var description: String
52: }

```

**Kod 31. Sadržaj RoadCondition objekta**

Funkcija `getRoadConditionFullReport()` nam vraća objekat tipa `RoadConditionDetails`, koji u suštini predstavlja objekat za sadržaj općih informacija na putevima, dok response od funkcije `getRoadConditions()` nam vraća objekat tipa `RoadCondition` koji predstavlja stanje na putu. Sadržaj navedenih funkcija je predstavljen u sekcijama Kod 29 i Kod 30, dok je sadržaj navedenih objekata predstavljen u sekcijama Kod 27 i Kod 31. Kada se podaci učitaju na mapu upotreba aplikacije je slična kao i za zaslon za prikaz radara. Na zaslonu od stanja na putevima također imamo opciju tj dugmad za

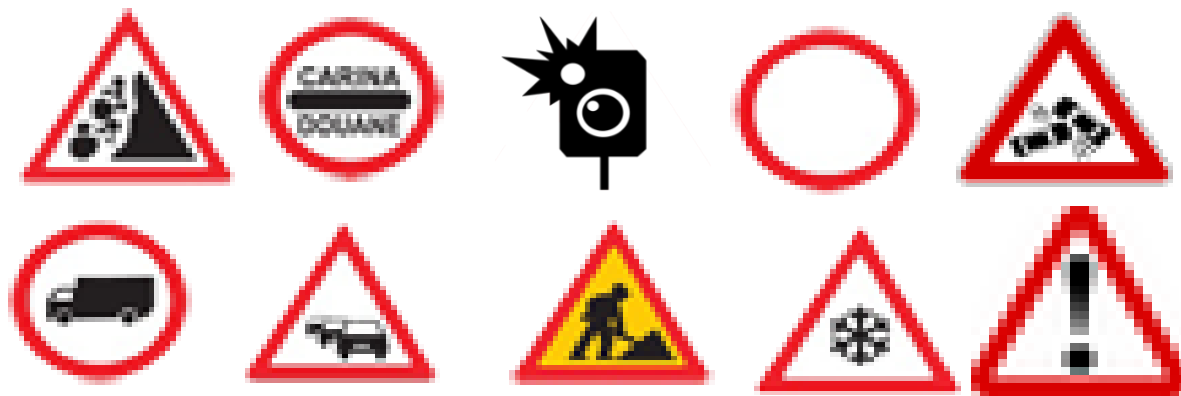


**Slika 19. Prikaz funkcionalnosti na zaslonu za prikaz stanja na putevima**

osvježivanje ili refresh podataka, promjenu tipa mape, prijavu novog stanja na putu, filtriranje stanja na putu u zavisnosti od samog stanja, te na ovom zaslonu imamo i opciju za prikaz detaljnih informacija o putevima.

Svako stanje je predstavljeno odgovarajućom slikom. Trenutno od stanja na putevima imamo mogućnosti prikazati sljedeća stanja:

- Granični prijelaz
- Sanacija kolovoza
- Potpuna obustava saobraćaja
- Zagušenje
- Zabrana za teretna vozila
- Odron
- Saobraćajna nezgoda
- Poledica
- Opasnost



**Slika 20. Prikaz svih znakova korištenih za prikaz stanja i radara na mapi**

Na slici 20 su prikazani svi znakovi i oznaka radara, koji su korišteni za prikaz stanja na mapi. Oznaka radara je kamera i ona se kroz kod direktno dodaje na `MKMarkerAnnotationView`, što možemo vidjeti u kodu sekciji 20, te u zavisnosti da li je Radar stacionarni ili najavljeni mi samo mijenjamo boju pozadine `MKMarkerAnnotationView-a`. Za stanje na putevima je drugačije i prikazano je u sekciji Kod 32, gdje mi uzimamo iz `RoadCondition`, uzimamo naslov slike koji dobijemo sa API-a i samo u zavisnosti od naziva koristimo odgovarajuću sličicu. Sve slike 20. su spremljene direktno u aplikaciju i mogu se pojaviti u bilo kojem trenutku na zaslonima ukoliko dobijemo objekat koji ih koristi.

```

12: class RoadConditionMarkerView: MKAnnotationView {
13:
14:     override var annotation: MKAnnotation? {
15:         willSet {
16:             guard let roadCondition = newValue as?
RoadCondition else { return }
17:             canShowCallout = true
18:             calloutOffset = CGPoint(x: 0, y: -2)
19:             detailCalloutAccessoryView =
getDescriptionLabel(text: roadCondition.subtitle)
20:             rightCalloutAccessoryView = UIButton(type:
.detailDisclosure)
21:             image = roadCondition.image
22:         }
23:     }
24:
25:     func getDescriptionLabel(text: String?) -> UILabel {
26:         let detailLabel = UILabel()
27:         detailLabel.numberOfLines = 2
28:         detailLabel.font = detailLabel.font.withSize(12)
29:         detailLabel.text = text
30:         return detailLabel
31:     }
32: }


```

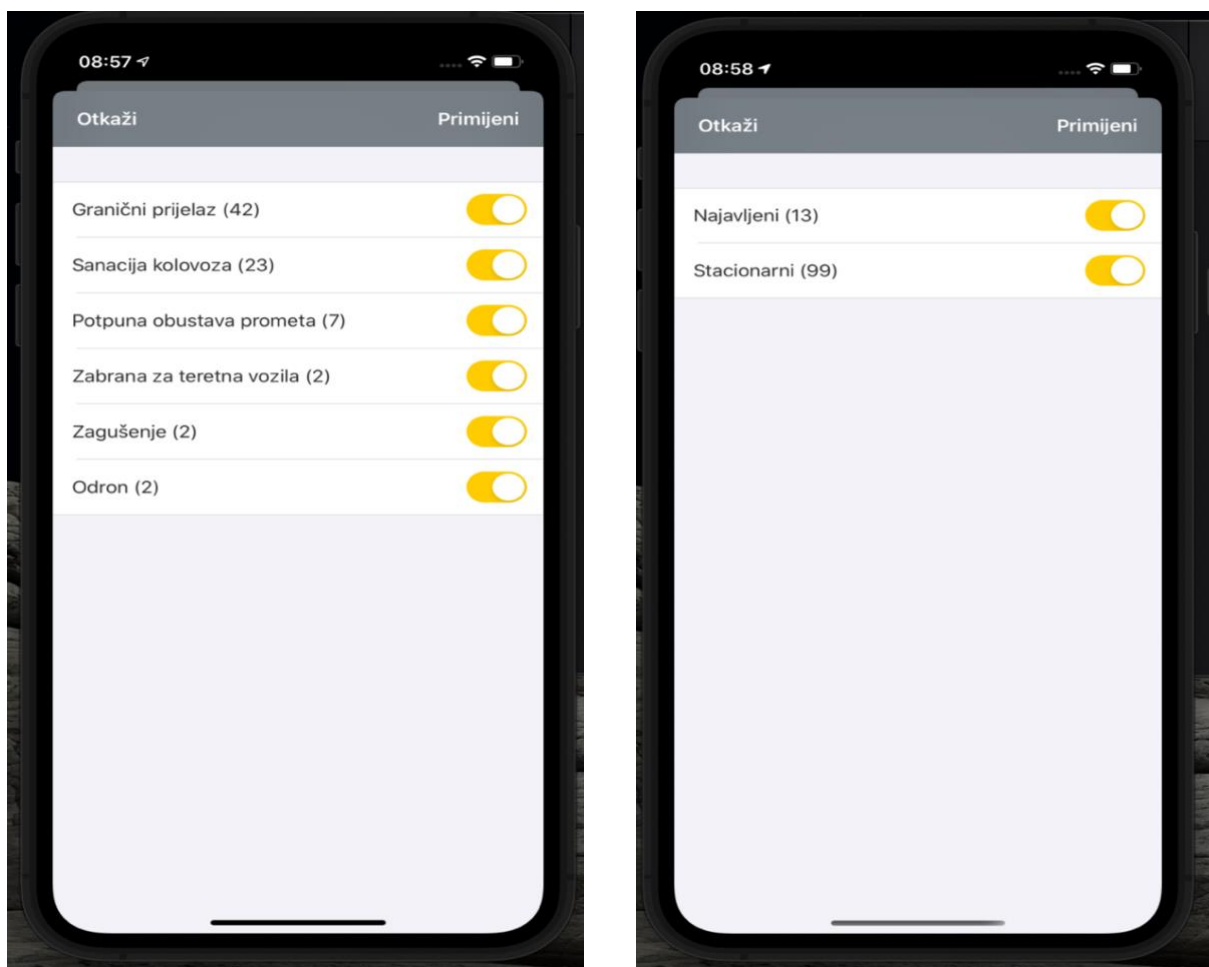
### Kod 32. Sadžaj RoadConditionMarkerView-a

RoadConditionMarkerView i RadarMarkerView su klase koje nasljeđuju MKMarkerAnnotationView klasu koji služi za vizuelnu reprezentaciju takozvanih oznaka na mapama. Ukoliko na zaslonima sa radarima ili stanjima na putevima odaberemo dugme refresh pozvati će se funkcija reloadScreen koja će postaviti sve vrijednosti na defaultne kao što je uključivanje loadera, isključivanje reload tj. refresh dugmeta jer ne želimo da korisnik pritisne više puta odjednom dugme, isključivanje dugmeta za filtriranje na oba zaslona i dugmeta za detaljne informacije na zaslonu za stanje na putevima. Pored toga sakivamo i oznaku koja se pojavljuje za prijavu novih radara ili stanja na putevima kada odaberemo opciju prijave, o tome ćemo u nastavku. Kada se to sve podese ponovo se pozove fetchData() tj da bi smo dohvatili podatke iz firestore baze. Na oba zaslona refreshScreen() funkcija se poziva i kada pošaljemo aplikaciju u pozadinu. U nastavku ćemo prvo objasniti zaslon za filtriranje, te zaslon za prikaz detaljnih informacija o radaru, stanju na putevima i općih informacija, te zaslon za prijavu novih radara i stanja na putevima.



## 5.4. Opis rada zaslona za filtriranje

Ova dva zaslona su po izgledu isti, razlike su u sadržaju. Kada sa zaslona za prikaz radara te zaslona za prikaz stanja na putevima pritisnemo na dugme  otvorit će nam se screen koji izgleda kao na Slici 21.




Slika 21. Zaslona za filtriranje stanja na putevima i zaslon za filtriranje radara

Do ovog zaslona možemo doći jedino u slučaju da imamo dva različita tipa oznake prikazana na mapi npr Najavljeni i Stacionarni radari. Filteri rade na način da ukoliko su uključeni te oznake koje su uključene biti će i prikazane na mapi.


Ukoliko isključimo neke od njih i pritisnemo Primijeni ili Apply dugme na vrhu ti podaci će se poslati našoj mapi te će na mapi biti prikazani samo određene oznake. Oznake možemo isključiti sve ali implementirano je da uvijek jedna mora biti uključena, to sam dodao iz razloga da ne bi korisnik se vratio na mapu i vidio praznu. Također ovaj zaslon se razlikuje od drugih jer se on prezentuje preko mape, a ne pusha predhodni zaslon u lijevu stranu. Ukoliko odaberemo opciju

Otkazi ili Cancel dugme, vraćamo se na mapu sa uključenim filterima koje smo imali uključene kada smo ušli na filter zaslon.

### 5.4.1. Implementacija zaslona filtriranje

Kao što sam naveo iznad do zaslona za filtriranje dolazimo kada sa zaslona za prikaz radara te zaslona za prikaz stanja na putevima pritisnemo na dugme . Prvo filter zaslon je u oba fajla spremljen kao `lazy var`. To je svojstvo čija se početna vrijednost ne izračunava do prvog poziva. To je dio obitelji svojstava u kojima imamo konstantna svojstva, izračunata svojstva i promjenjiva svojstva. Drugim riječima, to je lijena inicijalizacija.

Ovo je izvrstan način za optimizaciju našeg koda kako biste spriječili nepotreban posao. Razvrstavanje zbirke elemenata može biti skupo pa želimo biti sigurni da ovu operaciju izvodimo samo ako zapravo koristimo vrijednost. Stoga je lijeni var odlično rješenje. U Kod 33. Sekciji `FilterViewController.getViewController()` nam vraća Storyboard tj screen za prikaz radara.

Prije nego što se zaslon prikaze potrebno je da se učitaju vrijednosti tog zaslona i postavi dugme  na zaslonima da bude akzivno. Taj dio je urađen u sekciji Kod 24, isto vrijedi i za radare i za stanja na putevima, uz pomoć funkcije `setData(viewModel:FilterViewModel, filterType:FilterType)` koja postavlja viewModel i filter tip. ViewModel je uvijek isti dok tip može biti radars **ili** `roadConditions`, te u zavisnosti od njega definišemo određene stvari u viewModelu i viewControlleru za filtriranje.


```
68:     lazy var filterViewController: FilterViewController = {
69:         return FilterViewController.getViewController()
70:     }()
```

Kod 33. Prikaz inicijalizacije FilterScreena

Sadržaj funkcije setData je sljedeći

```
54: func setData(viewModel: FilterViewModel, filterType: FilterType) {
55:     self.viewModel = viewModel
56:     self.filterType = filterType
57:     if let view = contextView { view.reloadData() }
58: }
```

**Kod 34. Sadržaj setData funkcije unutar FilterViewControllera**

Kada smo postavili viewCotroller vrijeme je da ga i prikažemo, prikaz se vrši na klik  dugmeta. Te se izvršava sljedeći kod

```
216: @objc
217:     public func tapEditButton(_ sender: Any) {
218:         presentView(viewController: filterViewController)
219:     }
```

**Kod 35. Opis tapEditButton funkcije**

U funkciji iznad imamo prikaz da se filter zaslon prezentuje preko našeg glavnog zaslona. Sadržaj presentView funkcije je sljedeći

```
18: public func presentView(viewController: UIViewController) {
19:     navigationController?.present(viewController,
20:     animated: true, completion: nil)
21: }
```

**Kod 36. Sadržaj funkcije presentView**

Nakon toga se prikazuje filter screen nad našom mapom. Prilikom njegovog učitavanja pozivaju se dvije funkcije koje se nalaze u prilogu

```
68: extension FilterViewController: UITableViewDelegate, UITableViewDataSource {
69:     func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) ->
70:     Int {
71:         return viewModel.numberOfFilters
72:     }
73: func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) ->
74: UITableViewCell {
75:     let cell = tableView.dequeueReusableCell(with: FilterOptionTableViewCell.self,
76:     for: indexPath)
77:     cell.configureCell(radarOption: viewModel.getOption(index: indexPath.row))
78:     cell.optionSwitch.rx.isOn
79:     .changed
80:     .distinctUntilChanged()
81:     .bind { [weak self] isOn in
82:         guard let self = self else { return }
83:         self.viewModel.selectType(index: indexPath.row,
84:         selected: isOn)
85:         self.contextView.reloadData()
86:     }.disposed(by: cell.disposeBag)
87:     return cell
88: }
```

**Kod 37. Dvije glavne funkcije prilikom konfigurisanja tableView komponente**

U Kod 37. sekciji imamo dvije funkcije koje služe za konfiguriranje UITableView komponente. UITableView komponenta služi za prikaz listi na zaslonima. U toj sekciji vidimo da prva funkcija samo vraća broj filtera koje uzima iz viewModela koji smo ranije postavili. Taj broj nam definiše koliko će se puta pozvati druga funkcija. Druga funkcija služi za definisanje ćelije ili jednog objekta u listi. U našem slučaju to je FilterOptionTableViewCell ćelija čiji sadržaj koda je prikazan u nastavku

```
9: import UIKit
10: import RxSwift
11:
12: class FilterOptionTableViewCell: UITableViewCell {
13:
14:     @IBOutlet var optionName: UILabel!
15:     @IBOutlet var optionSwitch: UISwitch!
16:     var disposeBag = DisposeBag()
17:
18:     override func awakeFromNib() {
19:         super.awakeFromNib()
20:         clear()
21:     }
22:
23:     func configureCell(radarOption: TypeOption) {
24:         optionName.text = String(format: NUMBER_OF_TYPE,
25:                                   radarOption.typeValue,
26:                                   radarOption.numberOfElements)
27:         optionSwitch.isOn = radarOption.isOn
28:     }
29:
30:     override func prepareForReuse() {
31:         super.prepareForReuse()
32:         clear()
33:     }
34:
35:     private func clear() {
36:         disposeBag = DisposeBag()
37:         optionName.text = String()
38:         optionSwitch.isOn = false
39:         optionSwitch.isEnabled = true
40:     }
41: }
```

**Kod 38. Sadržaj FilterOptionTableViewCell ćelije**



**Slika 22. Grafički prikaz FilterOptionTableViewCell ćelije**

Unutar Kod 37. sekcije u drugoj funkciji prvo se inicijalizira navedena ćelija uz pomoć `dequeueReusableCell<T>(with type: T.Type, for indexPath: IndexPath) -> T` **where** `T: UITableViewCell` funkcije, te konfigurišemo ćeliju uz pomoć funkcije `configureCell(radarOption: TypeOption)` sa sadržajem koji dobivamo iz viewModela uz pomoć funkcije `getOption(index: Int) -> TypeOption`. `TypeOption` objekat je tipa protocol čiji sadržaj je prikazan u nastavku

```
11: protocol TypeOption {
12:     var numberOfElements: Int { get }
13:     var typeValue: String { get }
14:     var isOn: Bool { get }
15: }
```

**Kod 39. Sadržaj TypeOption protokola**

`numberOfElements` varijabla nam definiše koliko imamo elemenata tog tipa, `typeValue` naziv elementa za sortiranje, i `isOn` je opcija koja nam definiše da li je switch uključen ili ne. Sadržaj funkcije `configureCell(radarOption: TypeOption)` je prikazan u Kod 38. sekciji, gdje vidimo da ona postavlja sadržaj ćelije. Pored te funkcije unutar Kod 38. sekcije imamo još par funkcija kao što su

- `awakeFromNib()`
  - Priprema prijemnik za servis nakon što je učitao iz arhive Interface Builder -a ili datoteke nib.
- `prepareForReuse()`
  - Priprema ćeliju za višekratnu uporabu za ponovnu uporabu od strane delegata prikaza tablice.

Obje pozivaju `clear` funkciju koja postavlja sve vrijednosti labela i switcha na početne vrijednosti.

Nakon što smo izkonfigurisali ćeliju postavljamo u Kod 37. sekciji i prijemnik na switch tako da u slučaju da korisnik pritisne na switch ako je bio uključen on se isključi i obrnuto. Pri čemu se poziva odgovarajuća funkcija iz viewModela i ona izvršava tu promjenu. Nakon toga imamo poziv funkcije **self.contextView.reloadData()** koja kao što sam naziv njen govori ponovo reloada zaslona, tako da mi dobijemo osjećaj da smo kliknuli na switch I promijenili stanje sa on na off I obrnuto. Sadržaj viewModela nisam uključivao u ovaj rad ali može se pogledati u linku ispod za lokaciju na projekat koji se nalazi na GitHub stranici. Da bi ćelija bila samo prikazana da bi funkcija `dequeueReusableCell<T>(with type: T.Type, for indexPath: IndexPath) -> T` **where** `T: UITableViewCell` prošla bez grešaka potrebno je da registrujemo ćeliju prilikom učitavanja zaslona.

```

14:     @IBOutlet var navigationBarItem: UINavigationController! {
15:         didSet {
16:             navigationBarItem.leftBarButtonItem = cancelButton(action:
#selector(tapCancelButton))
17:             navigationBarItem.rightBarButtonItem = applyButton(action:
#selector(tapApplyButton))
18:         }
19:     }
20:
21:     @IBOutlet var contextView: UITableView! {
22:         didSet {
23:             contextView.registerCell(cellType: FilterOptionTableViewCell.self)
24:         }
25:     }
26:
27:     private var viewModel: FilterViewModel!
28:     private var filterType: FilterType = .radars
29:
30:     let filteredRadarsArray = PublishSubject<[Radar]>()
31:     let filteredRoadConditionsArray = PublishSubject<[RoadCondition]>()
32:
33:     override func viewDidLoad() { super.viewDidLoad() }
34:
35:     @objc
36:     private func tapCancelButton(_ sender: Any) {
37:         viewModel.resetFilters()
38:         contextView.reloadData()
39:         dismiss(animated: true)
40:     }
41:
42:     @objc
43:     private func tapApplyButton(_ sender: Any) {
44:         switch filterType {
45:             case .radars:
46:                 filteredRadarsArray.onNext(viewModel.filterRadars())
47:             case .roadConditions:
48:                 filteredRoadConditionsArray.onNext(viewModel.filterRoadConditions())
49:         }
50:
51:         dismiss(animated: true)
52:     }

```

**Kod 40. Ostali sadržaj FilterViewController-a**

Ukoliko je korisnik promijenio odgovarajuće filter opcije, on u svakom trenutku može odabrati cancel dugme koje se nalazi u lijevom ćošku ekrana ili Apply dugme koje se nalazi u desnom ćošku ekrana. Ukoliko pritisne Cancel filteri se resetuju na one vrijednosti koje su bile postavljene kada je korisnik ušao u filter zaslon, tj. izvršava se kod funkcije `tapCancelButton(_ sender: Any)` koja je definisana u Kod 40. sekciji.

Ukoliko korisnik odabere Apply dugme poziva se `tapApplyButton(_ sender: Any)` funkcija koja je definirana također u Kod 40. sekciji, te njen sadržaj jeste da u zavisnosti od tipa kojeg smo postavili prilikom definiranja samog viewControllera pošalje informaciju tj koji su filteri uključeni i isključeni preko odgovarajućeg pošiljatelja parent ViewController u kojem smo već prilikom njegove inicijalizacije definirali prijemnike na takve promjene. Primjeri pošiljatelja su definirani u Kod 40. sekciji. Dok su primjeri istih prijemnika za radare unutar RadarViewControllera i za stanja na putevima unutar RoadConditionViewControllera definisani u nastavku



```
RadarsMapViewController.swift
138: filterViewController.filteredRadarsArray.bind(onNext: { [unowned
self] radars in
139:         loadingIndicatorView.startAnimating()
140:         mapView.removeAnnotations(mapView.annotations)
141:         mapView.addAnnotations(radars)
142:         loadingIndicatorView.stopAnimating()
143:     })
144:     .disposed(by: disposeBag)

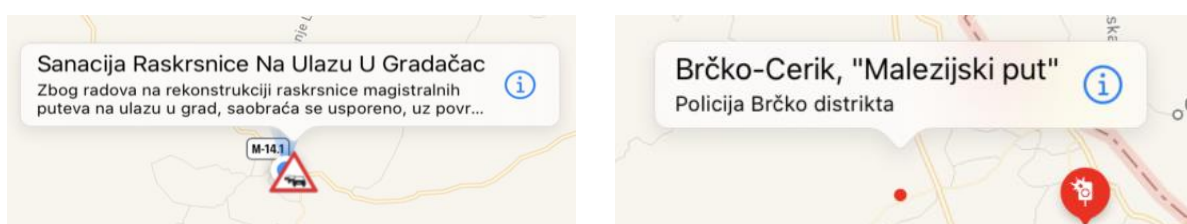
RoadConditionsViewController.swift
140: filterViewController.filteredRoadConditionsArray.bind(onNext: {
[unowned self] roadConditions in
141:         loadingIndicatorView.startAnimating()
142:         mapView.removeAnnotations(mapView.annotations)
143:         mapView.addAnnotations(roadConditions)
144:         loadingIndicatorView.stopAnimating()
145:     })
146:     .disposed(by: disposeBag)
```

#### Kod 41. Primjer inicijalizacije prijemnika promjena filtera

U Kod 41 sekciji vidimo da kada prijemnici prime odgovarajuće informacije od pošiljatelja, prvo se uključi loading zaslon, nakon toga se obrišu sve oznake na mapi i dodamo nove oznake koje želimo da su nam vidljive, te zaustavimo loading zaslon.

## 5.5. Opis rada zaslona za prikaz detalja

Do ovoga zaslona dolazimo na dva različita načina. Prvi način jeste klikom na dugme  na zaslonu za prikaz stanja na putevima. U ovom slučaju će nam biti prikazane opšte informacije o stanjima na putevima. To dugme nije vidljivo ukoliko nismo dobili nikakve informacije sa api strane. Drugi način jeste kada pritisnemo na  unutar kvadratića koji se pojavi kada pritisnemo na bilo koju oznaku stanja na putu ili radara na mapi.



Slika 23. Prikaz malog opisa stanja radara ili stanja na putu



Slika 24. Prikaz zaslona za prikaz detalja radara, stanja na putevima i općih informacija



Na zaslonu općih informacija imamo samo nazlov, podnaslov i detaljan opis. Na zaslonu za prikaz detaljnih informacija određenog radara imamo sljedeće

- Naslov
- Podnaslov
  - Inače predstavlja tip policijske uprave
    - MUP Zeničko-dobojskog kantona
    - MUP Kantona Sarajevo
    - MUP Unsko-sanskog kantona
    - MUP Tuzlanskog kantona
    - MUP Srednjobosanskog kantona
    - MUP Srednjobosanskog kantona
    - Policija Brčko distrikta
    - MUP Posavski
    - MUP Hercegovačko-neretvanskog kantona
    - MUP Bosansko-podrinjskog kantona
    - MUP Kantona 10 (Livno)
    - MUP Republike Srpske
- Tip radara
  - Stacionarni
  - Najavljeni
- Detaljan opis
- Vrijeme trajanja radara
- Broj prijava da radar nije aktivan

Za radare najvažnije je da objekat ima naslov i tip, sva ostala polja su opcionalna. Na zaslonu za prikaz detaljnih informacija stanja na putu imamo sljedeće

- Naslov
- Podnaslov
  - Inače predstavlja tip ulice na kojoj se nalazi stanje
    - Granični prijelaz
    - Autocesta
    - Magistralna cesta
    - Regionalna cesta
    - Gradska cesta

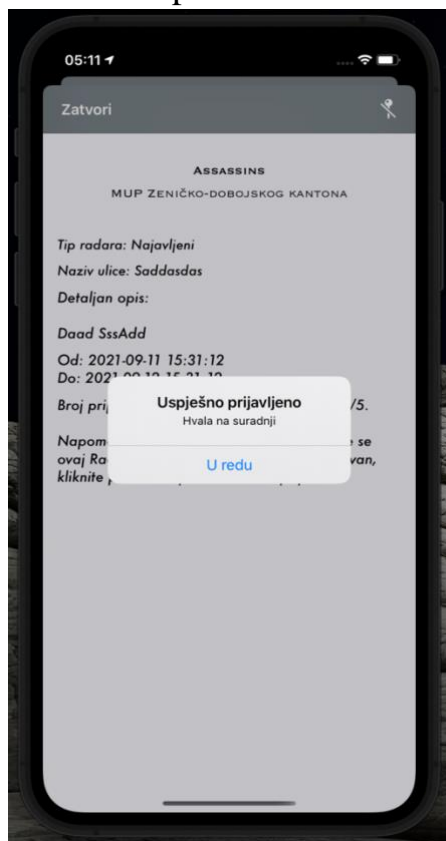
- Naziv ulice
- Detaljan opis
- Vrijeme trajanja stanja na putu
- Broj prijava da radar nije aktivan

Za stanja na putevima najvažnije je da objekat ima naslov, tip ulice i tip nesreće koji se ne prikazuje na zaslonu sa detaljima ali nam definiše koju ćemo sliku prikazati na mapi. Kao što možemo primijetiti na zaslonima sa detaljima radara i stanja na putu imamo i napomenu jednu. Napomena nam kaže da ukoliko smatramo da radar nije aktivan ili stanje na putu nije prisutno da možemo prijaviti




tako što ćemo pritisnuti na dugme u desnom ćošku novog zaslona. Klikom na ovo dugme mi šaljemo odgovarajući endpoint pri čemu povećavamo broj prijava unutar objekta. Na ekranu će se pojaviti odgovarajuća poruka da li je api poziv bio uspješan ili ne. Ukoliko API poziv jeste bio uspješan pritisnut ćemo na OK dugme i možemo vidjeti da je broj prijava povećan za jedan.

Ukoliko broj prijava je veći od 5 nakon prijave zaslon sa detaljima se zatvara i vršimo osvježivanje mape, te će navedeni radar koji smo prijavili ili stanje na putu biti izbrisano.



**Slika 25. Uspješno prijavljen neaktivan radar ili stanje na putu**

## 5.5.1. Implementacija zaslona za prikaz detalja

Sva tri načina otvaraju isti zaslon tj `DetailsViewController` razlika je samo u sadržaju koji se prikazuje na istom. Posmatrajmo slučaj kada se navedeni `ViewController` otvara ukoliko pritisnemo  dugme na zaslonu za prikaz stanja na putevima. Prvo da bi smo mogli pritisnuti na to dugme to dugme mora biti vidljivo njegova vidljivost se definiše unutar funkcije `prepareMapAndFilter(with roadConditions: [RoadCondition])` unutar `RoadConditionsViewController`-a. Sadržaj te funkcije je poprilično isti kao i u Kod 24. sekciji, ali unutar Kod 42. sekcije ćemo prikazati I tu verziju funkcije

```
180: private func prepareMapAndFilter(with roadConditions:
[RoadCondition]) {
181:     mapView.removeAnnotations(mapView.annotations)
182:     mapView.addAnnotations(roadConditions)
183:
184:     if let currentLocation = viewModel.userCurrentLocation
{
185:         mapView.setRegion(currentLocation, animated: true)
186:     }
187:
188:     if viewModel.getRoadConditionsDetails() != nil {
189:
navigationItem.rightBarButtonItemItems?.append(infoButton)
190:     }
191:
192:     let filterViewModel = FilterViewModel(roadConditions:
roadConditions)
193:     if filterViewModel.numberOfFilters > 1 {
194:         filterViewController.setData(viewModel:
filterViewModel, filterType: .roadConditions)
195:         let button = filterButton
196:         button.imageInsets = UIEdgeInsets(top: 0.0, left:
15, bottom: 0, right: -15)
197:         navigationItem.rightBarButtonItemItems?.append(button)
198:     }
199:
200:     reportButton.isHidden =
!Reachability.isConnectedToNetwork()
201:
202:     loadingIndicatorView.stopAnimating()
203:     delay(Constants.Time.TenSeconds) { [weak self] in
204:         guard let self = self else { return }
205:         self.reloadMapButton.isEnabled = true
206:     }
207: }
```

**Kod 42. Sadržaj `prepareMapAndFilter` funkcije unutar `RoadConditionViewControllera`**

Kao što možemo primijetiti unutar Kod 42. sekcije mi prikazujemo tzv. `infoButton` ako `getRoadConditionsDetails()` funkcija nam vrati neku vrijednost pri čemu ta vrijednost mora biti različita od `nil`. `Nil` je tzv. `null` vrijednost u drugim programskim jezicima. `getRoadConditionsDetails()` funkcija nam vraća `mainRoadConditionsDetails` gdje je ta vrijednost setovana unutar `handleRoadDetailsData()` funkcije koja se poziva samo u slučaju da smo izvršili promjenu vrijednosti unutar baze na mobitelu, u suprotnom ima vrijednost `nil`. Promjenu baze smo mogli izvršiti samo u slučaju da nam je `getRoadConditionFullReport()` funkcija iz Kod 29. Sekcije dohvatila


```
184: extension RoadConditionsViewModel:
NSFetchedResultsControllerDelegate {
185:
186:     private var mainRoadConditionsDetails:
RoadConditionDetails? = nil
187:     private func handleRoadDetailsData() {
188:         let roadDetails = roadInfoFRC.fetchedObjects ??
[]
189:         mainRoadConditionsDetails = roadDetails.first
190:     }
191:
192:     func getRoadConditionsDetails() ->
RoadConditionDetails? {
193:         return mainRoadConditionsDetails
194:     }
195:
196:     func controllerDidChangeContent(_ controller:
NSFetchedResultsController<NSFetchRequestResult>) {
197:         if controller == roadConditionFRC {
198:             handleRoadConditionData()
199:         } else if controller == roadInfoFRC {
200:             handleRoadDetailsData()
201:         }
202:     }
203: }
```

**Kod 43. Prikaz funkcija potrebnih za dohvaćanje opštih informacija iz baze mobitela**

neke vrijednosti sa API-a.

Da bi smo mogli slušati da li se događaju neke promjene u bazi mobitela potrebno je da naš `viewController` nasljeđuje `NSFetchedResultsControllerDelegate` te da ima implementiranu funkciju `controllerDidChangeContent(_ controller: NSFetchedResultsController<NSFetchRequestResult>)`.



Ako nam je  dugme vidljivo možemo pritisnuti na njega. Kada pritisnemo na to dugme okida se `tapInfoButton(_ sender: Any)` funkcija, unutar koje se poziva funkcija `presentView` koju smo objasnili u Kod 36. sekciji, kojoj se šalje vrijednost `DetailsViewController.showDetails(for: DetailsViewModel(roadDetails: roadDetails), delegate: self)`.

```
219: @objc
220: override func tapInfoButton(_ sender: Any) {
221:     guard let roadDetails = viewModel.getRoadConditionsDetails() else { return }
222:     presentView(viewController: DetailsViewController.showDetails(for:
DetailsViewModel(roadDetails: roadDetails),
223:                                     delegate: self))
224: }
```

#### Kod 44. Prikaz `tapInfoButton` funkcije

Kao što možemo primijetiti funkcija `tapInfoButton` je `overridan-a` tj. da ima neku defaultu vrijednost unutar **extension** `UINavigationController` gdje se nalaze sve `extenzije` pa sam i `infoButton`. Ekstezije nam služe da iste vrijednosti možemo koristiti na više različitih zaslona. Ovdje po prvi put se susrećemo sa `delegate` variablom. `Delegate` varijabla je u suštini `protocol` koji nam služi da bi smo mogli izvršiti neke operacije iz `child ViewController` u `parent ViewControlleru`.

Pored `delegate` mi u funkciju saljemo `DetailsViewModel` objekat kojem šaljemo naše `roadDetails`. `DetailsViewModel` ima tri različita konstruktora, tj da li se u `DetailsViewModel` šalje `roadDetails` ili `radar` ili `roadCondition` objekat, te na taj način konfiguriramo kako se prikazuju vrijednosti na zaslonu. Npr. na ovom zaslonu za opće informacije nemamo dugme za prijavu da su iste ne aktivne dok za radare i stanja na putevima imamo.

Kada se izvrši poziv funkcije `presentView` na našoj mapi se počinje učitavati zaslon sa detaljima. Zaslon sa detaljima je kao i zaslon za filtriranje ima na sebi `UITableView` komponentu i na nju registruje jednu ćeliju tipa `DetailsTableViewCell`. Broj ćelija na ovom zaslonu je 1 ne kao kod filtera. Ovo je urađeno da bi zaslon bio lakše skrollabilan, tj. da bi se mogli kretati kroz isti, u suprotnom bi tekst bio predug i korisnik ne bi mogao vidjeti cijele detalje.

```

98: extension DetailsViewController {
99:     static func getViewController() -> DetailsViewController {
100:         return UIStoryboard(name:
Constants.StoryboardIdentifiers.DetailsStoryboard, bundle: nil)
101:     }
102:     .instantiateViewControllerWithIdentifier(DetailsViewController.self)!
103: }
104: static func showDetails(for data: DetailsViewModel,
105:                         delegate: ViewProtocol?) ->
DetailsViewController {
106:     let detailsViewController =
DetailsViewController.getViewController()
107:     detailsViewController.setData(for: data, delegate:
delegate)
108:     return detailsViewController
109: }
110: }
111:
112: extension DetailsViewController: UITableViewDataSource,
UITableViewDelegate {
113:     func tableView(_ tableView: UITableView,
114:                   numberOfRowsInSection section: Int) -> Int {
115:         return 1
116:     }
117:
118:     func tableView(_ tableView: UITableView,
119:                   cellForRowAt indexPath: IndexPath) ->
UITableViewCell {
120:         let cell = tableView.dequeueReusableCell(with:
DetailsTableViewCell.self,
121:                                                  for: indexPath)
122:         cell.configure(for: data)
123:         return cell
124:     }
125: }

```

**Kod 45. Prikaz funkcija za učitavanje DetailsViewCotrollera i učitavanje ćelija na istom**

```

66: protocol ViewProtocol: AnyObject {
67:     func backButtonTaped()
68:     func reloadView()
69: }

```

**Kod 46. Prikaz sadržaj ViewProtocol-a**

```

11: enum DetailsType: String {
12:     case roadCondition
13:     case roadDetails
14:     case radar
15: }
16:
17: class DetailsViewModel {
18:
19:     var detailsType: DetailsType
20:     var dataId: String?
21:     var title: String?
22:     var subtitle: String?
23:     var road: String?
24:     var validFrom: String?
25:     var validTo: String?
26:     var text: String?
27:     var type: String?
28:     var numberOfDeletions: Int
29:     let manager: MainManager
30:
31:     init(roadCondition: RoadCondition,
32:         manager: MainManager = MainManager.shared) {
33:         self.detailsType = .roadCondition
34:         self.title = roadCondition.title
35:         self.subtitle = roadCondition.roadType
36:         self.road = roadCondition.road
37:         self.validFrom = roadCondition.validFrom
38:         self.validTo = roadCondition.validTo
39:         self.text = roadCondition.text
40:         self.numberOfDeletions = roadCondition.numberOfDeletions.intValue
41:         self.dataId = roadCondition.id
42:         self.manager = manager
43:     }
44:
45:     init(roadDetails: RoadConditionDetails,
46:         manager: MainManager = MainManager.shared) {
47:         self.detailsType = .roadDetails
48:         self.title = roadDetails.title
49:         self.subtitle = roadDetails.roadConditionType
50:         self.validFrom = roadDetails.validFrom
51:         self.validTo = roadDetails.validTo
52:         self.text = roadDetails.text
53:         self.numberOfDeletions = .zero
54:         self.dataId = roadDetails.id
55:         self.manager = manager
56:     }
57:
58:     init(radar: Radar,
59:         manager: MainManager = MainManager.shared) {
60:         self.detailsType = .radar
61:         self.title = radar.title
62:         self.subtitle = radar.policeDepartmentName
63:         self.validFrom = radar.validFrom
64:         self.validTo = radar.validTo
65:         self.road = radar.road
66:         self.text = radar.text
67:         self.type = radar.radarType.rawValue
68:         self.dataId = radar.id
69:         self.numberOfDeletions = radar.numberOfDeletions.intValue
70:         self.manager = manager
71:     }
72:
73:     func updateOrDeleteElement(_ completion: ((_ success: FirestoreStatus, _
error: String) -> Void)?) {
74:         manager.updateOrDelete(detailsViewModel: self, completion)
75:     }
76: }

```

**Kod 47. Sadržaj DetailsViewMoldela**

```

11: class DetailsTableViewCell: UITableViewCell {
12:
13:     @IBOutlet var titleLabel: UILabel!
14:     @IBOutlet var subtitleLabel: UILabel!
15:     @IBOutlet var roadLabel: UILabel!
16:     @IBOutlet var descriptionLabel: UILabel!
17:     @IBOutlet var durationLabel: UILabel!
18:     @IBOutlet var typeLabel: UILabel!
19:     @IBOutlet var deleteLabel: UILabel!
20:
21:     override func awakeFromNib() {
22:         super.awakeFromNib()
23:         clear()
24:     }
25:
26:     override func prepareForReuse() {
27:         super.prepareForReuse()
28:         clear()
29:     }
30:
31:     func configure(for data: DetailsViewModel) {
32:         titleLabel.text = data.title
33:
34:         if let subtitle = data.subtitle {
35:             subtitleLabel.text = subtitle
36:             subtitleLabel.isHidden = false
37:         }
38:
39:         if let road = data.road, road.isNotEmpty {
40:             roadLabel.text = String(format: STREET_NAME, road.withoutHtmlTags)
41:             roadLabel.isHidden = false
42:         }
43:
44:         if let text = data.text, text.isNotEmpty {
45:             descriptionLabel.text = String(format: DETAILS_DESCRIPTION,
46:                                             text.withoutHtmlTags)
47:             descriptionLabel.isHidden = false
48:         }
49:
50:         if let validFrom = data.validFrom, validFrom.isNotEmpty,
51:            let validTo = data.validTo.isNotNilNotEmpty ? data.validTo : NOT_DEFINED {
52:             durationLabel.text = String(format: DETAILS_DURATION, validFrom, validTo)
53:             durationLabel.isHidden = false
54:         }
55:
56:         if let type = data.type, type.isNotEmpty {
57:             typeLabel.text = String(format: RADAR_TYPE, type)
58:             typeLabel.isHidden = false
59:         }
60:
61:         if data.detailsType != .roadDetails {
62:             let detailsType = data.detailsType == .radar ? RADAR : ROAD_CONDITION
63:             deleteLabel.text = String(format: NUMBER_OF_REPORTS,
64:                                       detailsType,
65:                                       data.numberOfDeletions,
66:                                       detailsType)
67:             deleteLabel.isHidden = false
68:         }
69:     }
70:
71:     private func clear() {
72:         titleLabel.text = String()
73:         subtitleLabel.text = String()
74:         subtitleLabel.isHidden = true
75:         roadLabel.text = String()
76:         roadLabel.isHidden = true
77:         descriptionLabel.text = String()
78:         descriptionLabel.isHidden = true
79:         durationLabel.text = String()
80:         durationLabel.isHidden = true
81:         typeLabel.text = String()
82:         typeLabel.isHidden = true
83:         deleteLabel.text = String()
84:         deleteLabel.isHidden = true
85:     }
86: }
87:

```

**Kod 48. Sadržaj DetailsTableViewCell objekta**



Kao što možemo primijetiti unutar `DetailsViewController` objekta ukoliko neka od vrijednosti nije setovana došla sa API-a tu sekciju nećemo ni prikazati. Napisali smo ranije da ukoliko tip podatka za details zaslon nije opšte informacije tada na zaslonu imamo opciju da prijavimo da ako je radar ili stanje na putu pomaknuto ili otklonjeno. Za tu svrhu nam služi `tapReportButton(_ sender: Any)` funkcija koja je objašnjena ispod.

```
50: @objc
51: func tapReportButton(_ sender: Any) {
52:     let blurEffect = UIBlurEffect(style: .regular)
53:     let blurEffectView = UIVisualEffectView(effect: blurEffect)
54:     blurEffectView.frame = contextView.bounds
55:     blurEffectView.autoresizingMask = [.flexibleWidth,
    .flexibleHeight]
56:     contextView.addSubview(blurEffectView)
57:     loadingIndicatorView.startAnimating()
58:     data.updateOrDeleteElement { [weak self] status, message in
59:         guard let self = self else { return }
60:         switch status {
61:         case .updated:
62:             self.presentAlert(title: SUCCESSFULLY_REPORTED,
63:                               message: message,
64:                               buttonText: OK)
65:             for subview in self.contextView.subviews {
66:                 if subview is UIVisualEffectView {
67:                     subview.removeFromSuperview()
68:                 }
69:             }
70:             self.contextView.reloadData()
71:             self.delegate?.reloadView()
72:             self.loadingIndicatorView.stopAnimating()
73:         case .deleted:
74:             self.loadingIndicatorView.stopAnimating()
75:             for subview in self.contextView.subviews {
76:                 if subview is UIVisualEffectView {
77:                     subview.removeFromSuperview()
78:                 }
79:             }
80:             self.presentAlert(title: SUCCESSFULLY_DELETED,
81:                               message: message,
82:                               buttonText: OK,
83:                               handler: { _ in
84:                                   self.delegate?.reloadView()
85:                                   self.tapCloseButton(self)
86:                               })
87:         case .error:
88:             self.loadingIndicatorView.stopAnimating()
89:             self.contextView.removeFromSuperview()
90:             self.presentAlert(title: ERROR_DESCRIPTION,
91:                               message: message,
92:                               buttonText: OK)
93:         }
94:     }
95: }
```

**Kod 49. Sadržaj `tapReportButton` funkcije unutar zaslona za detalje**

Unutar Kod 49. sekcije možemo primijetiti da kada pritisnemo na report dugme prvo se pokaže loader, te se pozove funkcija `updateOrDeleteElement(_ completion: ((_ success: FirestoreStatus, _ error: String) -> Void)?)` koja poziva API poziv za update ili delete nad određenim objektom. Nakon toga ako su API pozivi bili uspješni ili ne prikazujemo popup koji nam ispisiuje odgovarajuću poruku. Ukoliko je API poziv bio uspješan mi prikažemo popup I reloadamo screen. Ukoliko je broj prijava bio veći od 5 tada se za API poziv koristi delete, te ako on bude uspješan prikazujemo popup te korisnik pritisne OK zaslon za detalje se zatvara I mi samo osvježimo našu mapu.

```

358: func deleteElement(elementId: String?,
359:                     type: DetailsType,
360:                     _ completion: ((_ success: FirestoreStatus, _ error:
String) -> Void)? = nil) {
361:     guard let elementId = elementId else { return }
362:     switch type {
363:     case .roadCondition:
364:         firestoreDataBase.collection("RoadConditions").document(elementId).delete() { err
in
365:             if let err = err {
366:                 completion?(.error, err.localizedDescription)
367:             } else {
368:                 completion?(.deleted, THANKS)
369:             }
370:         }
371:     case .radar:
372:         firestoreDataBase.collection("Radars").document(elementId).delete() { err in
373:             if let err = err {
374:                 completion?(.error, err.localizedDescription)
375:             } else {
376:                 completion?(.deleted, THANKS)
377:             }
378:         }
379:     case .roadDetails:
380:         break
381:     }
382: }
383:
384: func updateOrDelete(detailsViewModel: DetailsViewModel,
385:                     _ completion: ((_ success: FirestoreStatus, _ error:
String) -> Void)?) {
386:     guard let id = detailsViewModel.dataId else {
387:         completion?(.error, WRONG_ID)
388:         return
389:     }
390:     detailsViewModel.numberOfDeletions += 1
391:
392:     if detailsViewModel.numberOfDeletions >= 5 {
393:         deleteElement(elementId: id, type: detailsViewModel.detailsType,
completion)
394:     } else {
395:         updateVisibility(id: id, detailsViewModel: detailsViewModel,
completion)
396:     }
397: }

```

**Kod 50. Sadržaj deleteElement i updateOrDelete funkcija**

```

399: func updateVisibility(id: String,
400:                        detailsViewModel: DetailsViewModel,
401:                        _ completion: ((_ success:
FirestoreStatus, _ error: String) -> Void)?) {
402:
403:     switch detailsViewModel.detailsType {
404:     case .roadCondition:
405:
406:         firestoreDataBase.collection("RoadConditions").document(id).setData([
407:             "numberOfDeletions":
detailsViewModel.numberOfDeletions,
408:             ], merge: true) { err in
409:                 if let err = err {
410:                     completion?(.error, err.localizedDescription)
411:                 } else {
412:                     completion?(.updated, THANKS)
413:                 }
414:             case .radar:
415:
416:         firestoreDataBase.collection("Radars").document(id).setData([
417:             "numberOfDeletions":
detailsViewModel.numberOfDeletions,
418:             ], merge: true) { err in
419:                 if let err = err {
420:                     completion?(.error, err.localizedDescription)
421:                 } else {
422:                     completion?(.updated, THANKS)
423:                 }
424:             case .roadDetails:
425:                 break
426:             }
427:         }

```

**Kod 51. Sadržaj updateVisibility funkcije**

```

RoadConditionsViewController.swift
256: extension RoadConditionsViewController: MKMapViewDelegate {
257:
258:     func mapView(_ mapView: MKMapView, annotationView view:
MKAnnotationView, calloutAccessoryControlTapped control: UIControl) {
259:         guard let roadCondition = view.annotation as? RoadCondition else {
260:             return }
261:         presentView(viewController: DetailsViewController.showDetails(for:
DetailsViewModel(roadCondition: roadCondition),
262:         delegate: self))
263:     }


RadarsMapViewController.swift
242: extension RadarsMapViewController: MKMapViewDelegate {
243:
244:     func mapView(_ mapView: MKMapView, annotationView view:
MKAnnotationView, calloutAccessoryControlTapped control: UIControl) {
245:         guard let radar = view.annotation as? Radar else { return }
246:         setReportPinVisibility()
247:         presentView(viewController: DetailsViewController.showDetails(for:
DetailsViewModel(radar: radar), delegate: self))
248:     }
249: }

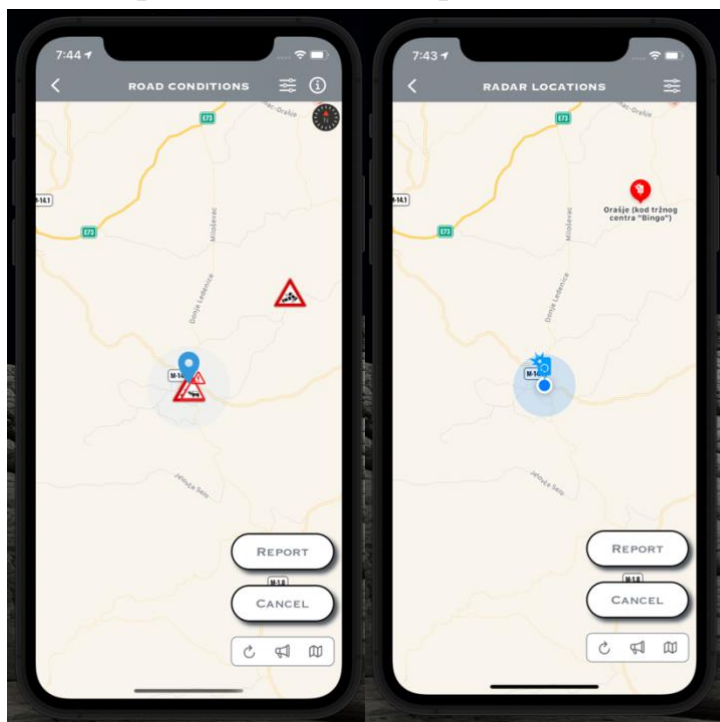
```

**Kod 52. Implementacija otvaranja DetailsViewControllera sa oznake**

U Kod 52. sekciji možemo uočiti na koji način smo realizovali otvaranje DetailsViewCotrollera iz oznake radara ili stanja na putu, tj bilo je potrebno da naši viewConrolleri nasljeđuju MKMapViewDelegate tj da imaju implementiranu funkciju **func** mapView(\_ mapView: MKMapView, annotationView view: MKAnnotationView, calloutAccessoryControlTapped control: UIControl) koja reguliše svaki klik na bilo koju od oznaka na mapi koje su prikazane bili to radari ili stanja na putu.

## 5.6. Opis rada zaslona za prijavu novih radara ili stanja na putevima

Kao što sam naveo ranije korisnik u našoj aplikaciji ima i mogućnost da prijavi nova stanja na putevima ili novi radar. Klikom na  dugme koje je prisutno na zaslonima za prikaz radara i stanja na putevima ukoliko je korisnik priključen na Internet, na mapi se pojavljuje jedna oznaka na mapi iznad naše trenutne lokacije. Krećući se po mapi horizontalno ili vertikalno mi pozicioniramo navedenu oznaku na odgovarajuće mjesto. Pored oznake na mapi se pojavljuju dva dugmeta tj. dugme Report i dugme Cancel, takav prikaz imamo na Slici 26. Klikom na Cancel dugme smatra se kao da smo odustali od prijave radara ili stanja na putu, te se nakon toga navedena oznaka i dugmad uklone sa mape. Ukoliko korisnik pritisne na dume Report, korisniku se na zaslonu pojavljuje novi zaslon



sa odgovarajućim poljima. Zajedničko na ova dva zaslona koji su prikazani na Slici 27. jeste lokacija radara ili stanja na putu, naziv radara ili stanja na putu, ulica i detaljan opis. Lokacija predstavlja poziciju gdje smo ostavili našu oznaku sa prethodnog zaslona. Za naziv radara ili stanja na putu potrebno je unijeti neki kratki opis stanja. Za polje Ulica unosimo naziv ulice, dok detaljni opis sama riječ nam govori na šta se odnosi

Slika 26. Pozicija oznake za prijavu

Pored toga imamo polje Radar type ili Road Type. Pri čemu Radar Type može biti Stacionarni ili Najavljeni, dok Road Type može biti

- Autocesta
- Granični prijelaz
- Gradska cesta
- Magistralna cesta
- Regionalna cesta

Pored toga na radarima imamo polje Police department koje ima opcije

- MUP Zeničko-dobojskog kantona
- MUP Kantona Sarajevo
- MUP Unsko-sanskog kantona
- MUP Tuzlanskog kantona
- MUP Srednjobosanskog kantona
- MUP Srednjobosanskog kantona
- Policija Brčko distrikta
- MUP Posavski
- MUP Hercegovačko-neretvanskog kantona
- MUP Bosansko-podrinjskog kantona
- MUP Kantona 10 (Livno)
- MUP Republike Srpske

dok na stanjima na putevima imamo polje Type of road condition koje ima opcije

- Granični prijelaz
- Sanacija kolovoza
- Potpuna obustava prometa
- Zabrana za teretna vozila
- Zagušenje
- Odron
- Saobraćajna nezgoda
- Poledica
- Opasnost

koji ujedno predstavljaju ikone znakova koje možemo prikazati za stanje na putu.

Da bi korisnik mogao prijaviti radar potrebno je da unese tip radara i naziv, dok da bi prijavio stanje na putu potrebno je da odabere tip ulice, naziv i tip stanja na putu.

Ukoliko korisnik ne unese obavezna polja a pritisne na Report dugme koje se nalazi na dnu ekrana korisnik će dobiti odgovarajuću poruku u vidu popUp-a da je potrebno da unese obavezna polja. Kada korisnik unese obavezna polja onda kada pritisne na report dugme, pokreće se odgovarajući API poziv te korisnik dobiva poruku na ekranu da li je prijava bila uspješna ili ne. Ukoliko jeste bila uspješna zaslon za prijavu se zatvara, pokreće se spinner i osvježavamo odgovarajuću mapu za prikaz tj dohvatamo sa API-a stanja na putevima ili radare što smo objasnili na samom početku ovog rada.

The image displays two mobile application screens side-by-side, both showing report forms. The left screen is titled "REPORT RADAR" and the right screen is titled "ROAD CONDITION REPORT". Both screens have a similar layout with the following fields and elements:

- Location field:** Contains the coordinates "44.87725999999998, 18.425141600000018".
- Type field:** A dropdown menu with the text "Select radar type" (left) or "Select road type" (right).
- Title field:** A text input field with the placeholder "Insert radar title" (left) or "Insert title for the road condition" (right).
- Street field:** A text input field with the placeholder "Street".
- Police department field:** A dropdown menu with the text "Select police department" (left) or "Select condition sign" (right).
- Description field:** A text input field with the placeholder "Detailed description".
- Report button:** A button at the bottom with the text "REPORT RADAR" (left) or "ROAD CONDITION REPORT" (right).

Slika 27. Prikaz report zaslona

## 5.6.1. Implementacija zaslona za prijavu radara ili stanja na putu

Kao što smo već rekli do navedenog zaslona dolazimo prvo klikom na dugme



Već prilikom same inicijalizacije zaslona za prikaz radara i stanja na putevima mi smo postavili odgovarajuće pretplatnike na klik za ovo dugme a i za dugmad Report i Cancel koji se pojave kada stisnemo na već navedeno dugme. To smo za oba zaslona i za prikaz radara i za stanja na putevima uradili unutar `setupObservers()` funkcije. U ovom trenutku ćemo se fokusirati na njih u navedenom kodu

```
157: reportButton.rx.tap.bind { [unowned self] in
158:         setReportPinVisibility(isVisible: true)
159:     }.disposed(by: disposeBag)
160:
161: cancelReportButton.rx.tap.bind { [unowned self] in
162:         setReportPinVisibility()
163:     }.disposed(by: disposeBag)
164:
165: confirmReportButton.rx.tap.bind { [unowned self] in
166:     pushViewController(viewController:
ReportViewController.showReportPage(for:
viewModel.getCenterLocation(for: mapView), reportType:
.radarReport, delegate: self))
167:     }.disposed(by: disposeBag)
```

Kod 54. Prikaz pretplatnika na dugmad za reporting



Kao što možemo primijetiti prvo kada pritisnemo na \_\_\_\_\_ dugme poziva se `setReportPinVisibility(isVisible: Bool = false)` funkcija

```
217: private func setReportPinVisibility(isVisible: Bool = false) {
218:     reportContainer.isHidden = !isVisible
219:     radarPin.isHidden = !isVisible
220:
221:     guard
222:         isVisible == true,
223:         let userCurrentLocation =
viewModel.userCurrentLocation
224:     else { return }
225:     mapView.setRegion(userCurrentLocation, animated: true)
226: }
```

Kod 53. Sadržaj `setReportVisibility` funkcije

koju smo već ranije vidali u kod sekcijama. Sadržaj iste je prikazan u nastavku

U funkciji vidimo da ona utiče na vidljivost Report i Cancel dugmadi i odgovarajućeg pina na ekranu koji su vidljivi na Slici 27. Prvo znači prikažemo taj sadržaj. Ukoliko korisnik pritisne Cancel dugme u Kod 53. sekciji vidimo da pozivamo funkciju `setReportVisibility()` bez parametara što znači da ćemo sakriti dugmad i pin sa ekrana.

Ukoliko pritisnemo na Report dugme u Kod 53. sekciji vidimo da pozivamo `pushViewController()` funkciju koja je objašnjena u Kod 9. sekciji gdje njoj prosljeđujemo `viewController` tipa `ReportViewController`

tako što pozivamo `showReportPage(for location: CLLocation, reportType: ReportType, delegate: ViewProtocol? = nil)` funkciju.

Vidimo da funkcija uzima lokaciju tj lokacija gdje se nalazi sama oznaka, `reportType` koji može biti `radarReport` ili `roadConditionReport` u zavisnosti da li prijavljujemo radar ili stanje na putu, te delegate čiju smo svrhu objašnjavali ranije tj u Kod 46. sekciji.

Kada se zaslon počne inicijalizirati možemo primijetiti da se na njemu nalazi `UITableView` te da ima samo jednu ćeliju koja se prikazuje na ekranu, tj u zavisnosti od `reportType` mi prikazujemo `RadarReportTableViewCell` ili `RoadConditionReportTableViewCell`, to možemo primijetiti u Kod 55. sekciji. Ćelije su poprilično iste razlika jeste u sadržaju ali krenimo redom. na vrhu svake ćelije nalazi se zapis lokacije na kojoj smo ostavili pin ili oznaku na mapi, nakon toga imamo polja za unos naziva radara ili stanja na putu, polje za unos naziva ulice te polje za detaljni opis. Također na dnu obje ćelije imamo dugme za prijavu radara ili stanja na putu. Na oba zaslona imamo po dvije takozvane `Picker` komponente koje imaju odgovarajuće elemente koji su predloženi nama da mi ne unosimo ručno, te komponente na `RadarReportTableViewCell`

- Komponenta za odabir tipa radara
- Komponenta za odabir policiske uprave

dok za `RoadConditionReportTableViewCell` imamo

- Komponenta za odabir tipa ulice
- Komponenta za odabir tipa stanja na putu

Njihove opcije su već navedene na početku ovog poglavlja. Kao što sam ranije rekao korisnik mora iz poštovati tj. unijeti obavezna polja da bi prijavio radar ili stanje na putu `handleApplyButton()` funkcije čiji sadržaj prikazan u Kod 56 i 57 sekcijama. Kada su ispunjeni svi uslovi poziva se funkcija `addNewRadar()` te api poziv koji se isto naziva za radare, dok se za stanje na putevima poziva `addNewRoadCondition()`. Unutar njih prosljeđujemo i `completion` objekat, tj okvir u kojem se nalazi dio koda koji će se izvršiti nakon što se api poziv završi bilo li on uspješan ili ne, a tu ujedno se vraćamo na zaslon sa mapom



```

109: extension ReportViewController: UITableViewDataSource, UITableViewDelegate {
110:     func tableView(_ tableView: UITableView, numberOfRowsInSection section:
Int) -> Int {
111:         return 1
112:     }
113:
114:     func tableView(_ tableView: UITableView, cellForRowAt indexPath:
IndexPath) -> UITableViewCell {
115:         switch reportType {
116:             case .radarReport:
117:                 let cell = tableView.dequeueReusableCell(with:
RadarReportTableViewCell.self, for: indexPath)
118:                 cell.configure(viewModel: RadarReportCellViewModel(for:
location))
119:                 cell.messageTransmitter.bind(onNext: { [unowned self] adviser in
120:                     showAlert(title: adviser.title,
121:                             message: adviser.message,
122:                             buttonTitle: OK,
123:                             handler: adviser.isError ? { _ in
124:                                 delegate?.reloadView()
125:                                 tapBackButton(self)
126:                             } : nil)
127:                 })
128:                 .disposed(by: disposeBag)
129:
130:                 cell.loaderStatus.bind(onNext: { [unowned self] status in
131:                     if status == true {
132:                         loadingIndicatorView.startAnimating()
133:                     } else {
134:                         loadingIndicatorView.stopAnimating()
135:                     }
136:                 })
137:                 .disposed(by: disposeBag)
138:                 return cell
139:             case .roadConditionReport:
140:                 let cell = tableView.dequeueReusableCell(with:
RoadConditionReportTableViewCell.self, for: indexPath)
141:                 cell.configure(viewModel: RoadConditionReportCellViewModel(for:
location))
142:                 cell.messageTransmitter.bind(onNext: { [unowned self] adviser in
143:                     showAlert(title: adviser.title,
144:                             message: adviser.message,
145:                             buttonTitle: OK,
146:                             handler: adviser.isError ? { _ in
147:                                 delegate?.reloadView()
148:                                 tapBackButton(self)
149:                             } : nil)
150:                 })
151:                 .disposed(by: disposeBag)
152:
153:                 cell.loaderStatus.bind(onNext: { [unowned self] status in
154:                     if status == true {
155:                         loadingIndicatorView.startAnimating()
156:                     } else {
157:                         loadingIndicatorView.stopAnimating()
158:                     }
159:                 })
160:                 .disposed(by: disposeBag)
161:                 return cell
162:             case .none:
163:                 return UITableViewCell()
164:         }
165:     }
166: }

```

**Kod 55. Učitavanje ćelija na UITableView unutar ReportViewControllera**

```

99: private func handleApplyButton() {
100:     if let roadType = viewModel.getRoadType(for:
roadTypePickerField.text) {
101:         if let conditionType =
viewModel.getConditionType(for: conditionTypePickerField.text) {
102:             if let title =
roadConditionTitleContext.text, title.isNotEmpty {
103:                 loaderStatus.onNext(true)
104:                 viewModel.addNewRoadCondition(roadType:
roadType,
105:                                             road:
roadConditionStreetContext.text,
106:                                             text:
roadConditionDetailContext.text,
107:                                             title:
title,
108: conditionType: conditionType) { [weak self] status,error in
109: self?.messageTransmitter.onNext(Adviser(title:
ROAD_CONDITIONS_INFO, message: error, isError: true))
110: self?.loaderStatus.onNext(false)
111:     }
112:     } else {
113:         messageTransmitter.onNext(Adviser(title:
ERROR_DESCRIPTION, message: ENTER_ROAD_CONDITION_TITLE))
114:         loaderStatus.onNext(false)
115:         return
116:     }
117:     } else {
118:         messageTransmitter.onNext(Adviser(title:
ERROR_DESCRIPTION, message: SELECT_CONDITION_SIGN))
119:         loaderStatus.onNext(false)
120:         return
121:     }
122:     } else {
123:         messageTransmitter.onNext(Adviser(title:
ERROR_DESCRIPTION, message: SELECT_ROAD_TYPE))
124:         loaderStatus.onNext(false)
125:         return
126:     }
127: }

```

**Kod 56. Sadržaj handleApplyButton unutar ReporRoadConditionCell**

```

95: private func handleApplyButton() {
96:     if let radarType = viewModel.getRadarType(for:
radarTypePickerField.text) {
97:         if let title = radarTitleContext.text, title.isNotEmpty {
98:             loaderStatus.onNext(true)
99:             viewModel.addNewRadar(policeDepartmentName:
MUPTYPEPickerField.text,
100:                                 road: radarStreetContext.text,
101:                                 text: radarDetailContext.text,
102:                                 title: title,
103:                                 type: radarType) { [weak self]
status,error in
104:                 self?.messageTransmitter.onNext(Adviser(title:
RADARS_INFO, message: error, isError: true))
105:                 self?.loaderStatus.onNext(false)
106:             }
107:         } else {
108:             messageTransmitter.onNext(Adviser(title: ERROR_DESCRIPTION,
message: ENTER_RADAR_TITLE))
109:             loaderStatus.onNext(false)
110:             return
111:         }
112:     } else {
113:         messageTransmitter.onNext(Adviser(title: ERROR_DESCRIPTION,
message: SELECT_RADAR_TYPE))
114:         loaderStatus.onNext(false)
115:         return
116:     }
117: } File:
/Users/eldar.haseljic/Desktop/ios/BosniaRoadTraffic/BosniaRoadTraffic/Common/Rep
ortScreen/RadarReportCell/RadarReportViewCell.swift
95: private func handleApplyButton() {
96:     if let radarType = viewModel.getRadarType(for:
radarTypePickerField.text) {
97:         if let title = radarTitleContext.text, title.isNotEmpty {
98:             loaderStatus.onNext(true)
99:             viewModel.addNewRadar(policeDepartmentName:
MUPTYPEPickerField.text,
100:                                 road: radarStreetContext.text,
101:                                 text: radarDetailContext.text,
102:                                 title: title,
103:                                 type: radarType) { [weak self]
status,error in
104:                 self?.messageTransmitter.onNext(Adviser(title:
RADARS_INFO, message: error, isError: true))
105:                 self?.loaderStatus.onNext(false)
106:             }
107:         } else {
108:             messageTransmitter.onNext(Adviser(title: ERROR_DESCRIPTION,
message: ENTER_RADAR_TITLE))
109:             loaderStatus.onNext(false)
110:             return
111:         }
112:     } else {
113:         messageTransmitter.onNext(Adviser(title: ERROR_DESCRIPTION,
message: SELECT_RADAR_TYPE))
114:         loaderStatus.onNext(false)
115:         return
116:     }
117: }

```

**Kod 57. Sadržaj handleApplyButton unutar ReporRoadConditionCell**

```

RoadConditionReportCellViewModel.swift
71: func addNewRoadCondition(roadType: RoadType,
72:                           road: String?,
73:                           text: String?,
74:                           title: String,
75:                           conditionType: ConditionType,
76:                           _ completion: ((_ success: Bool, _ error:
String) -> Void)?) {
77:     let dateFormatter : DateFormatter = DateFormatter()
78:     dateFormatter.dateFormat = "yyyy-MM-dd HH:mm:ss"
79:     let currentDate = Date()
80:     let startDate = dateFormatter.string(from: currentDate)
81:     let endDate = dateFormatter.string(from:
currentDate.dateByAddingDays(1))
82:     let roadConditionParameters: RoadConditionParameters =
83:         RoadConditionParameters(icon: conditionType.icon, coordinates:
locationString,
84:                                 road: road, text: text, title:
title, validFrom: startDate,
85:                                 validTo: endDate, roadTypeID:
roadType.id, roadTypeName: roadType.name,
86:                                 updatedAt: startDate)
87:     manager.addNewRoadCondition(roadConditions:
roadConditionParameters, completion)
88: }

RadarReportCellViewModel.swift
72: func addNewRadar(policeDepartmentName: String?, road: String?, text:
String?, title: String, type: RadarType, _ completion: ((_ success: Bool, _
error: String) -> Void)?) {
73:     let dateFormatter : DateFormatter = DateFormatter()
74:     dateFormatter.dateFormat = "yyyy-MM-dd HH:mm:ss"
75:     let currentDate = Date()
76:     let startDate = dateFormatter.string(from: currentDate)
77:     var endDate: String? = nil
78:
79:     if type == .announced {
80:         endDate = dateFormatter.string(from:
currentDate.dateByAddingDays(1))
81:     }
82:
83:     var radarParameters: RadarParameters =
RadarParameters(coordinates: locationString,
84:                                                         title:
title,
85:                                                         type:
type.pickerView)
86:
87:     if policeDepartmentName != SELECT_POLICE_DEPARTMENT {
88:         radarParameters.policeDepartmentID =
getPoliceDepartmentID(for: policeDepartmentName)
89:         radarParameters.policeDepartmentName = policeDepartmentName
90:     }
91:
92:     radarParameters.road = road
93:     radarParameters.text = text
94:     radarParameters.validFrom = startDate
95:     radarParameters.validTo = endDate
96:     radarParameters.updatedAt = startDate
97:     manager.addNewRadar(radarParameters: radarParameters, completion)
98: }

```

**Kod 58. Sadržaj addNewRadar i addNewRoadCondition funkcija u viewModel-ima**

```

304: func addNewRadar(radarParameters: RadarParameters,
305:                 _ completion: ((_ success: Bool, _ error: String) ->
Void)?) {
306:     let ref = firestoreDataBase.collection("Radars")
307:     let docId = "RadarID-\(ref.document().documentID)"
308:
309:     firestoreDataBase.collection("Radars").document(docId).setData([
310:         "id": docId,
311:         "title": radarParameters.title ,
312:         "coordinates": radarParameters.coordinates,
313:         "type": radarParameters.type,
314:         "road": radarParameters.road ?? NSNull(),
315:         "valid_from": radarParameters.validFrom ?? NSNull(),
316:         "valid_to": radarParameters.validTo ?? NSNull(),
317:         "text": radarParameters.text ?? NSNull(),
318:         "numberOfDeletions": 0,
319:         "category_id": radarParameters.policeDepartmentID ?? NSNull(),
320:         "category_name": radarParameters.policeDepartmentName ?? NSNull(),
321:         "updated_at": radarParameters.updatedAt ?? NSNull()
322:     ], merge: true) { err in
323:         if let err = err {
324:             completion?(false, err.localizedDescription)
325:         } else {
326:             completion?(true, RADAR_SUCCESSFULLY_REPORTED)
327:         }
328:     }
329: }
330:
331: func addNewRoadCondition(roadConditions: RoadConditionParameters,
332:                         _ completion: ((_ success: Bool, _ error: String)
-> Void)?) {
333:     let ref = firestoreDataBase.collection("RoadConditions")
334:     let docId = "RoadConditionID-\(ref.document().documentID)"
335:
336:     firestoreDataBase.collection("RoadConditions").document(docId).setData([
337:         "id": docId,
338:         "title": roadConditions.title ,
339:         "coordinates": roadConditions.coordinates,
340:         "icon": roadConditions.icon,
341:         "road": roadConditions.road ?? NSNull(),
342:         "valid_from": roadConditions.validFrom ?? NSNull(),
343:         "valid_to": roadConditions.validTo ?? NSNull(),
344:         "text": roadConditions.text ?? NSNull(),
345:         "category_id": roadConditions.roadTypeID,
346:         "numberOfDeletions": 0,
347:         "category_name": roadConditions.roadTypeName,
348:         "updated_at": roadConditions.updatedAt ?? NSNull()
349:     ], merge: true) { err in
350:         if let err = err {
351:             completion?(false, err.localizedDescription)
352:         } else {
353:             completion?(true, ROAD_CONDITION_SUCCESSFULLY_REPORTED)
354:         }
355:     }
356: }

```

#### Kod 59. Sadržaj addNewRadar i addNewRoadCondition unutar mainManagera

U Kod 59. sekciji vidimo da u obje funkcije nad objektom `firestoreDataBase` pozivamo metod `collection`, gdje pristupamo odgovarajućoj kolekciji objekata. Unutar te kolekcije kreiramo `document` sa odgovarajućim `Id` em te postavimo njegove podatke i ako je taj proces uspješan ili ne pozovemo `completion` koji smo ranije spomenuli, prikazujemo korisniku odgovarajući `popup` te se vraćamo na zaslon sa mapom.

## 5.7. Unos podataka u firebase uz pomoć skripti

Prilikom izrade ovog diplomskog rada za unos podataka u bazu napisane su odgovarajuće JavaScript skripte, koje nisam dodavao u ovaj diplomski rad ali se nalaze u github firebase folderu. Za pokretanje isti potrebno je da imamo Node.js i NPM instaliran na našem uređaji. Node.js je okruženje za izvršavanje koje uključuje sve što vam je potrebno za izvršavanje programa napisanog u JavaScript-u. NPM označava Node Package Manager, aplikaciju i spremište za razvoj i dijeljenje JavaScript koda. Da bi pokrenuli odgovarajuću skriptu potrebno je da pozovemo komandu

```
node addMainRoadConditionsInfo.js roadConditions16_30.json
node addRoadConditionsToFirebase.js roadConditionsAll.json
node addRadarsToFirestore.js radars.json
node deleteAllRadars.js
node deleteAllRoadConditions.js
node deleteExpiredRadars.js
```

**Kod 60. Načini pokretanja skripti za unos i brisanje podataka**

Također potrebno je da instaliramo odgovarajuće module kao što su date-and-time i firebase na našem uređaji

## 6. Zaključak

Prilikom izrade ovog diplomskog rada sam unaprijedio svoje znanje prilikom korištenja odgovarajućih Framework-a, reaktivnog programiranja te samog korištenja FireBase komponenti. Aplikacija je veoma zanimljiva i ima puno opcija, naravno da bi bila idealna mora ispuniti još neke uslove ali za izradu ovoga diplomskog rada je iz poštovala sve kriterije. Aplikaciju je moguće koristiti na tri različita jezika Engleski, Njemački i Bosanski. Aplikacija neće biti na Apple store zato što nemam Account na istoj stranici ali je moguće isto postaviti ukoliko neka firma ili organizacija to odluči uz određene dorade na aplikaciji najvjerojatnije.

## 7. Literatura

1. <https://github.com/eldarhaseljic/IOS-BosniaRoadTraffic>
2. <https://nodejs.org/en/>
3. <https://www.raywenderlich.com/7738344-mapkit-tutorial-getting-started>
4. <https://firebase.google.com/>
5. <https://developer.apple.com/xcode/>
6. <https://www.raywenderlich.com/1228891-getting-started-with-rxswift-and-rxcocoa>
7. <https://github.com/objcio/core-data/blob/master/SharedCode/NSManagedObjectContext%2BExtensions.swift>
8. <https://stackoverflow.com/questions/26056062/uiviewcontroller-extension-to-instantiate-from-storyboard>
9. <https://stackoverflow.com/questions/55964080/register-and-dequeue-uitableviewcell-of-specific-type>
10. <https://stackoverflow.com/questions/24034544/dispatch-after-gcd-in-swift>
11. <https://www.hackingwithswift.com/example-code/language/how-to-make-array-access-safer-using-a-custom-subscript>
12. <https://gist.github.com/cozzin/d6c45b51905d56f0bc8b097bb6aa4ce8>
13. <https://stackoverflow.com/questions/39348729/core-data-viewcontext-not-receiving-updates-from-newbackgroundcontext-with-nsf>
14. <https://stackoverflow.com/questions/30743408/check-for-internet-connection-with-swift/39782859#39782859>
15. <https://github.com/objcio/core-data/blob/master/SharedCode/Managed.swift>
16. <https://bihamk.ba/>