Mobilna aplikacija za praćenje informacija o koronavirusu

Juhasz, Josip

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

MOBILNA APLIKACIJA ZA PRAĆENJE INFORMACIJA O KORONAVIRUSU

Diplomski rad

Josip Juhasz

Osijek, 2022.

Sadržaj

1. UVOD

Krajem prosinca 2019. godine zabilježen je prvi slučaj SARS-CoV-2 virusa u gradu Wuhanu (Kina). Virus, koji je uzrok nove bolesti dišnih puteva pod nazivom COVID-19, također je i uzrok pandemije koronavirusa koju je Svjetska zdravstvena organizacija (eng. *WHO – World Health Organisation*) proglasila 11. ožujka 2020. godine. Pandemija je imala velike posljedice na svjetsko gospodarstvo, turizam i općenito na svakodnevni život ljudi u svijetu. S obzirom na to da se virus prenosi socijalnim kontaktom i da se brzo širi, uvedene su brojne epidemiološke mjere protiv njegovog širenja – zatvorene su državne granice, odgođeni su javni događaji, ograničene su nastavne aktivnosti i zaustavljen je rad trgovačkih centara. Iako su uvedene brojne epidemiološke mjere, u travnju 2020. godine je zabilježeno preko milijun slučajeva.

Mobilni uređaji su dio ljudske svakodnevice. Njihova prenosivost, funkcionalnost i dostupnost, omogućila je da na svijetu postoji oko 5 milijardi korisnika mobilnih uređaja, a većina tih uređaja su tzv. pametni uređaji (eng. *smartphones*). Korištenje fotoaparata, GPS uređaja, čitanje najnovijih vijesti i mogućnost kupovanja preko interneta samo su neke od stvari koje aplikacije pametnih uređaja omogućuju. Kako su mobilne aplikacije dostupne većini ljudi na svijetu, tako je i glavni zadatak ovog rada izrada mobilne aplikacije koja će korisniku omogućavati praćenje slučajeva koronavirusa, čitanje najnovijih vijesti vezane za pandemiju i skeniranje COVID digitalnih potvrda. Korisniku će biti omogućeno praćenje broja potvrđenih, aktivnih, oporavljenih i umrlih slučajeva pojedine države i cijeloga svijeta uzrokovanih COVID-19, prikaz najnovijih vijesti o slučajevima koronavirusa i epidemiološkim mjerama, te prikaz detalja COVID digitalnih potvrda nakon skeniranja.

U drugom poglavlju opisane su aplikacije koje su slične aplikaciji ovog rada, te njihove razlike. Alati i tehnologije koji su korišteni prilikom izrade rada, predstavljeni su u trećem poglavlju. Četvrto poglavlje sadrži programsko rješenje kojim je izrađena mobilna aplikacija korištenjem različitih tehnologija i alata. Poglavlje sadrži i primjere zaslona koji prikazuju izgled same aplikacije.

2. PREGLED PODRUČJA TEME

Na tržištu aplikacija postoji nekoliko aplikacija koje su slične aplikaciji ovoga rada. Najpoznatija aplikacija je CovidGO dostupna na [1]. CovidGO je mobilna aplikacija koja omogućava validaciju QR kodova na EU digitalnim COVID potvrdama izdanim u Republici Hrvatskoj i državama članica EU. Razlika između aplikacija je u tome što CovidGO aplikacija nema mogućnost informiranja korisnika o trenutnom broju slučaja koronavirusa u svijetu. CovidGO aplikacija ne sadrži informacije o najnovijim vijestima vezanim za koronavirus. Aplikacija slična CovidGO aplikaciji je Corona-Warn-App izrađena od strane Robert Koch instituta [2]. Aplikacija omogućuje skeniranje COVID digitalnih potvrda, te obavještavanje korisnika o opasnosti ako je korisnik u blizini zaražene osobe koja je također korisnik aplikacije. STOP-COVID 19, dostupna na [3], predstavlja mobilnu aplikaciju u kojoj su prikazane upute o mjerama zaštite kako bi se spriječio rast slučajeva koronavirusa, te omogućuje obavijest ostalim članovima aplikacije o zarazi. Aplikacija je kao i CovidGO osnovana u Hrvatskoj. NHS-Covid-19 je mobilna aplikacija koja sadrži informacije o simptomima koronavirusa, dokumentaciju o cjepivima protiv koronavirusa i upute koje korisnik treba pratiti u slučaju zaraze virusom [4]. COVID Tracker Ireland je aplikacija koja korisnicima pruža informacije o slučajevima koronavirusa u Irskoj [5]. Mnoge aplikacije na tržištu, kao i COVID Tracker aplikacija, omogućuju pristup informacijama vezanim za određenu državu. Zbog neodržavanja web-servera s kojih se dohvaćaju informacije o slučajevima koronavirusa, mnoge aplikacije su zaustavljene zbog nedovoljno informacija s kojima bi mogle obavještavati korisnike. Mobilna aplikacija ovog rada kombinira više web-servera na kojima se nalaze podaci o koronavirusu kako bi korisnicima pružila što točnije i ispravnije informacije.

Commented [JB1]: Dodajte u literaturu link na App Store aplikacije COvidGO

3. TEHNOLOGIJE I ALATI KORIŠTENI U IZRADI RADA

3.1. Swift programski jezik

Swift je objektno orijentirani programski jezik koji se koristi za pisanje softverskih programa za Apple proizvode. Predstavljen je 2014. godine na WWDC-u (*Apple Worldwide Developers Conference*) za razvijanje macOS, watchOS, tvOs i iOS uređaja uz već postojeći Objective-C programski jezik [6].

Commented [JB2]: Poziv na reference mora biti unutar rečenice prije točke

3.1.1. Tipovi podataka, konstante i varijable

Tablica 3.1. prikazuje tipove podataka koji se mogu koristiti u Swift programskom jeziku.

Commented [JB3]: Tablica ne smije biti šira od teksta

Commented [JJ4R3]:

Tip podatka	Opis
Int	Koristi se za prikaz cijelih brojeva
Double	Koristi se za prikaz 64-bitnih decimalnih brojeva
Float	Koristi se za prikaz 32-bitnih decimalnih brojeva
Character	Koriste za prikaz isključivo jednog znaka (simbola).
String	Koristi se za prikaz skupine znakova.
Bool	Može sadržavati samo dvije vrijednosti – istina(*true*) ili laž (*false*).
Neobavezan	Može sadržavati omotanu vrijednost ili vrijednost može biti odsutna
Enumeracija	Tip podatka za grupu povezanih vrijednosti

Tab. 3.1. Tipovi podataka u Swift programskom jeziku

Konstante i varijable se prije korištenja moraju deklarirati. Konstante se deklariraju s *let*, dok se varijable deklariraju s *var*. Vrijednost konstanti se ne može promijeniti jednom kada je postavljena, dok se vrijednost varijable može promijeniti u daljnjem pisanju programa. Programski kod 3.1 prikazuje primjer deklariranja konstante i varijable [6].

```
import UIKit

let constant = 1
var variable = 2

variable = 3

print("Constant value - \(constant)")
print("Variable value - \(variable)")

// Constant value - 1
// Variable value - 3
```

Programski kod 3.1. Primjer deklariranje varijable i konstante

Konstantama i varijablama nije potrebno definirati tip podatka, ali Swift programski jezik nudi da se pokraj imena varijable ili konstante definira tip podatka kako bi čitanje programskog koda bilo lakše. Programski kod 3.2 prikazuje primjere deklariranja konstante i varijable s definiranim tipom podatka [6].

```
import UIKit

let constant: String = "Constant"
var variable: String = "Empty"

variable = "Variable"

print("Constant value - \(constant)")
print("Variable value - \(variable)")

// Constant value - Constant
// Variable value - Variable
```

Programski kod 3.2. Primjeri konstante i varijable s definiranim tipom podatka

Programer može stvoriti novo ime za postojeće tipove podataka korištenjem ključne riječi *typealias*. *Typealias* ne stvara nove tipove podataka nego samo daje novo ime postojećem tipu podatka. *Typealias* može zamijeniti ime bilo kojeg tipa podatka. Primjer *typealiasa* prikazan je na programskom kodu 3.3.

```
import UIKit

typealias newIntName = Int

let intConstant: newIntName = 4

print("Constant value - \(intConstant)")

// Constant value - 4
```

Programski kod 3.3. Primjer *typealiasa*

Swift nudi korištenje neobaveznog (eng. *optional*) u situacijama u kojima postoji mogućnost da vrijednost varijable bude odsutna. Prema [6] neobavezan može imati dvije mogućnosti: varijabla sadrži vrijednost i ona je umotana (eng. *wrapped*) ili varijabla ne sadrži vrijednost (*nil*). Kako bi se varijabla označila kao neobavezan, potrebno je dodati upitnik (?) pokraj tipa podatka varijable. Programski kod 3.4 prikazuje primjere neobaveznog.

```
import UIKit

let firstOptional: String? = "Hello, world!"
let secondOptional: String? = nil

print("First optional value - \(firstOptional)")
print("Second optional value - \(secondOptional)")

// First optional value - Optional("Hello, world!")
// Second optional value - nil
```

Commented [JB5]: Fontovi na programskim kodovima moraju biti približno iste veličine

Programski kod 3.4. Primjeri neobaveznog

Programski kod 3.4 prikazuje dva primjera neobaveznog. U drugom primjeru vrijednost varijable je odsutna (*nil*). Prvi primjer sadrži vrijednost *Hello, world!* koja je umotana i koju je potrebno odmotati da vrijednost više ne bude neobavezan. Swift nudi četiri načina za odmotavanje vrijednosti neobaveznog. Prvi način je korištenje prisilnog odmotavanja (eng. *forced unwrapping*). Ovaj način nije prikladan za korištenje jer ako je vrijednost varijable *nil*, doći će do rušenja aplikacije. Prisilno odmotavanje koristi se samo u slučajevima u kojima je programer siguran da će varijabla sadržavati vrijednost. Kako bi se koristilo prisilno odmotavanje, potrebno je dodati uskličnik (!) pokraj neobaveznog. Programski kod 3.5 prikazuje korištenje prisilnog odmotavanja [6].

```
import UIKit

let optionalExample: String?

optionalExample = "Hello, world!"

print("Optional value - \(optionalExample!)")

// Optional value - Hello, world!
```

Programski kod 3.5. Korištenje prisilnog odmotavanja neobaveznog

Prema [6] vezivanje neobaveznog (eng. *optional binding*) je način odmotavanja neobaveznog koji je vrlo sličan *if-else* kontroli toka. Ako neobavezan nije *nil,* odmotana vrijednost je dodijeljena novoj konstanti i daljnje operacije se mogu izvršavati pomoću konstante. Kako bi se vrijednost odmotala, koristi se *if-let* izraz. Programski 3.6 prikazuje primjer korištenja odmotavanja vezivanjem neobaveznog.

Commented [JB6]: Prevelika praznina ispod

```
import UIKit

let optionalExample: String?

optionalExample = "Hello, world!

if let optionalExample = optionalExample {
   print("Optional value - \(optionalExample)“)
} else {
   print("Value is nil")
}
// Optional value - Hello, world!
```

Programski kod 3.6. Odmotavanje korištenjem vezivanja neobaveznog

Nil spajanje (eng. *nil coalescing*) omogućuje, ako je vrijednost *nil,* postavljanje zadane vrijednosti. Kako bi se ovaj način koristio, potrebno je dodati dva znaka upitnika (??) pokraj vrijednosti koja sadrži umotanu vrijednost. *Nil* spajanje uz vezivanje neobaveznog predstavlja najčešće i najsigurnije načine odmotavanja vrijednosti neobaveznog. Programski kod 3.7 prikazuje primjer korištenja *nil* spajanja [6].

```
import UIKit

let optionalExample: String? = nil

print("Optional value- \(optionalExample ?? "Hello, world!")")
// Optional value - Hello, world!
```

Programski kod 3.7. Primjer odmotavanja korištenjem *nil* spajanja

Ulančavanje neobaveznog (eng. *optional chaining*) omogućeno je korištenjem upitnika (?) pokraj vrijednosti koja se želi odmotati. Vrlo je slično prisilnom odmotavanju. Glavna razlika je, ako je vrijednost *nil*, da će se prilikom prisilnog odmotavanja aplikacija srušiti dok kod ulančavanja neće. Programski kod 3.8 prikazuje primjer korištenja neobaveznog ulančavanja [6].

```
import UIKit

class User {
    let name: String?

    init(name: String){
        self.name = name
    }
}

class Car {
    var owner: User?

    init(){
        self.owner = nil
    }
}

let user = User(name: "Joe")
var car: Car?
car = Car()
car?.owner = user

if let car = car, let user = car.owner, let name = user.name {
    print("Name is - \(name)")
}
// Name is - Joe
```

Programski kod 3.8. Primjer korištenja neobaveznog ulančavanja

Prema [6] enumeracija (eng. *enumeration*) predstavlja zajednički tip podatka za grupu povezanih vrijednosti i omogućuje rad s tim vrijednostima. Slučajevi (eng. *cases*) u enumeracijama ne moraju sadržavati vrijednosti. Ako slučajevi sadrže vrijednosti one mogu biti bilo kojeg tipa podatka. Enumeracije su često korištene u svojstvima klasa i struktura. Enumeracije mogu sadržavati vlastite inicijalizatore, mogu implementirati protokole i mogu biti proširene kako bi se proširila njihova funkcionalnost. Primjer enumeracije prikazan je programskim kodom 3.9.

Commented [JB7]: Popraviti to u cijelom radu!!

```
import UIKit

enum Car {
   case bmw
   case audi
   case renault
}

let car = Car.bmw

switch car {
   case .bmw:
      print("BMW")
   case .audi:
      print("Audi")
   case .renault:
      print("Audi")
}
// BMW
```

Programski kod 3.9. Primjer enumeracije

3.1.2. Programska funkcija

Funkcija predstavlja izdvojenu programsku cjelinu koja izvršava određene programske naredbe. Programer može napisati svoju vlastitu funkciju, ali može koristiti i funkcije koje su pohranjene u razne biblioteke. Funkciju je, poput varijabli, potrebno deklarirati. Osim naziva funkcije, funkciji je moguće dodati tipove podataka koji se nazivaju argumenti. Funkcija može kao rezultat izvršavanja programskih naredbi vratiti određeni tip podatka, ali isto tako funkcija ne mora vraćati nikakvu vrijednost, odnosno u tom slučaju funkcija vraća Void. Programski kod 3.10 prikazuje primjer funkcije [7].

```
import UIKit

func sumNumbers(firstNumber: Int, secondNumber: Int) -> Int {
   return firstNumber + secondNumber
}

let sum = sumNumbers(firstNumber: 2, secondNumber: 3)
print("Sum is \(sum)")
// Sum is 5"
```

Programski kod 3.10. Primjer funkcije koja zbraja dva cjelobrojna broja

3.1.3. Zatvoreni izrazi u programiranju

Prema [8] zatvoreni izraz (eng. *closure*) je samostalni blok funkcionalnosti koji se može proslijediti i koristiti na različitim mjestima u kodu. Zatvoreni izraz je jedna od najkorištenijih stvari u Swift programiranju jer može pohraniti reference bilo kojoj varijabli ili konstanti iz konteksta u kojima su definirane. Vrlo su slični funkcijama jer također mogu primati parametre i imati povratnu vrijednost. Funkcija uvijek mora imati ime, dok zatvoreni izraz ne. Zatvoreni izraz može biti globalan i ugniježđen. Najkorišteniji predefinirani zatvoreni izrazi u Swift programskom jeziku su *map* i *sorted.* Primjeri korištenja zatvorenog izraza prikazani su programskim kodom 3.11.

```
import UIKit

let firstClosureExample = {
   print("Hello, world!")
}

let secondClosureExample: (String) -> () = { name in
   print("Name is \(name)")
}
firstClosureExample()
secondClosureExample("Joe")
// Hello, world!
// Name is Joe
```

Programski kod 3.11. Primjeri korištenja zatvaranih izraza

3.1.4. Strukture i klase

Strukture i klase predstavljaju fleksibilne konstrukcije opće namjene koje postaju građevni blokovi programa. Strukture i klase sadrže svojstva i metode koje predstavljaju njihove funkcionalnosti. Svojstva su varijable i konstante koje pohranjuju vrijednosti, dok su metode funkcije. Klasama i strukturama je potrebno definirati inicijalizatore kako bi se postavila početna vrijednost njihovih svojstava. Osim što mogu implementirati protokole za pružanje standardne funkcionalnosti određene vrste mogu biti i proširene kako bi se proširila njihova funkcionalnost osim zadane implementacije. Klase predstavljaju referentne vrste, dok su strukture vrijednosne vrste. Klase za razliku od struktura mogu nasljeđivati neku drugu klasu. Programski kod 3.12 prikazuje primjer strukture i klase [9].

```
import UIKit

class Owner {
   let name: String

   init(name: String){
      self.name = name
   }

   func getName() -> String {
      return "Name is \(name)"
   }
}

struct Car {
   var owner: Owner

   init(owner: Owner){
      self.owner = owner
      print("Car owner is \(owner.name)")
   }
}

let owner = Owner(name: "Joe")
let car = Car(owner: owner)
print(owner.getName())
// Car owner is Joe
// Name is Joe
```

Programski kod 3.12. Primjer strukure i klase

3.1.5. Protokol u programiranju

Prema [10] protokol predstavlja skup metoda, svojstava i drugih zahtjeva koji odgovaraju dijelu funkcionalnosti ili određenom zadatku. Protokol može biti korišten od strane klase, strukture ili enumeracije kako bi bila osigurana implementacija tih zahtjeva. Svaki tip koji zadovoljava zahtjeve protokola kažemo da on implementira protokol. Osim specificiranja zahtjeva koji moraju biti implementirani od strane tipa koji implementira protokol, protokol se može i proširiti kako bi se ostvarila dodatna funkcionalnost. Unutar protokola se ne definira funkcionalnost funkcije ili nekog svojstva. Njihova funkcionalnost se definira u klasama, strukturama ili enumeracijama. Programski kod 3.13 prikazuje primjer protokola.

```
import UIKit

protocol CarProtocol {
   func getCarDetails()
}

class Car: CarProtocol {
   let name: String
   let color: String

   init(name: String, color: String){
      self.name = name
      self.color = color
   }

   func getCarDetails() {
      print("Car name is \(name) and car color is \(color)")
   }
}

let car = Car(name: "BMW", color: "Black")
car.getCarDetails()
// Car name is BMW and car color is Black
```

Programski kod 3.13. Primjer protokola

3.2. SwiftUI radni okvir

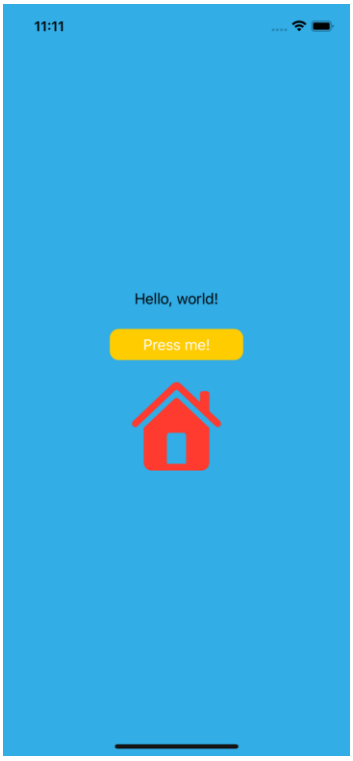
Apple je 2019. godine predstavio novi radni okvir pod nazivom SwiftUI. SwiftUI je jednostavan i inovativan način za izgradnju korisničkog sučelja na svim Apple platformama koristeći samo jedan skup alata i API-ja (*Application Programming Interface*). SwiftUI omogućuje programeru izradu korisničkog sučelja na deklarativni način, odnosno opisivanje izgleda korisničkog sučelja i reagiranje na promjene stanja. Deklarativni način omogućuje lakši način čitanja i shvaćanja koda. SwiftUI se bazira na korištenju pogleda (eng. *view*). Pogled je tip koji predstavlja dio korisničkog sučelja i sadrži modifikatore koji služe za njegovu konfiguraciju. Prilikom promjene stanja nekog pogleda, pogled se renderira i prikazuje se njegovo novo promijenjeno stanje. Okvir sadrži sve što je potrebno za izradu korisničkog sučelja kao što je tekst, lista, gumb, slika i još mnogo ostalih komponenti [11].

3.2.1. Elementi korisničkog sučelja

Prema [11] SwiftUI nudi velik broj elemenata korisničkog sučelja koji su bili dostupni i u UIKit-u. Svaki element korisničkog sučelja u SwiftUI-u predstavlja pogled koji se može oblikovati raznim modifikatorima. Najkorišteniji elementi korisničkog sučelja su:

- *Text* predstavlja pogled koji omogućuje prikaz jedne ili više linija samo za čitanje teksta. Pomoću modifikatora moguće je promijeniti svojstva teksta poput veličine i fonta.
- *Button* je pogled koji omogućuje izvođenje neke operacije. Kao parametar prima operaciju koje će se izvršiti pritiskom na *Button* i elemente korisničkog sučelja s kojima će *Button* biti dizajniran.
- *Image* predstavlja pogled koji prikazuje sliku. Pomoću *frame* modifikatora moguće je postaviti veličinu slike.
- *List* je spremnik koji predstavlja retke podataka poredanih u jednom stupcu pružajući mogućnost odabira jednog ili više članova. Kao parametar može primiti niz nekih podataka.
- *TextField* je pogled koji prikazuje tekstualno sučelje koje se može uređivati.
- *HStack* je pogled koji slaže poglede u vodoravnu liniju. Za razliku od *LazyHStacka* koji renderira poglede kada trebaju prikazani na zaslonu, *HStack* renderira sve poglede odjednom.
- *VStack* je pogled koji slaže poglede u okomitu liniju. *LazyVStack* renderira poglede kada ih je potrebno prikazati, dok *VStack* renderira sve poglede odjednom.

Slika 3.1. prikazuje primjer aplikacije koja sadrži osnovne elemente korisničkog sučelja.



Sl. 3.1. Primjer elemenata korisničkog sučelja

3.2.2. Omotači svojstava

Prema [12], kako je SwiftUI deklarativni radni okvir, umjesto korištenja delegata ili ostalih obrazaca za upravljanje stanjem koji se obično koriste u imperativnim radnim okvirima kao što je UIKit, koriste se omotači svojstava (eng. *property wrappers*). Najkorišteniji omotači svojstava u Swift programskom jeziku su :

- *@Binding* omogućuje deklariranje svojstva koji će sadržavati vrijednost svojstva iz drugog pogleda i vrijednost tog svojstva će biti prikazana i korištena u oba pogleda.

- *@StateObject* predstavlja omotač koji se koristi kada je potrebno stvoriti referencu unutar jednog od pogleda i ako dođe do promjene nekog svojstva pogled se renderira. Implementira *ObservableObject* protokol koji omogućuje emitiranje prije promjene objekta.
- *@ObservedObject* je omotač koji omogućuje da pogledi mogu promatrati stanje nekog vanjskog objekta i da mogu biti obaviješteni prilikom njegove promjene. Sličan je *@StateObjectu*, te je bitna razlika da *@ObservedObject* objekti ne budu stvoreni u istom pogledu jer postoji mogućnost da ih SwiftUI uništi.
- *@Environment* se koristi za rad sa SwiftUI predefiniranim vrijednostima kao što su omogućivanje rada uređaja u svijetlom i tamnom načinu, dohvaćanje vrijednosti piksela zaslona i dohvaćanje trenutne faze zaslona.
- *@Published* je jedan od najkorištenijih omotača i omogućuje stvaranje objekata koji omogućuju reagiranje na njihove promjene. SwiftUI automatski promatra promjene i ako dođe do njih, dolazi do renderiranja svih pogleda koji sadržavaju taj objekt. Ako se svojstvo nekog objekta, koje je označeno *@Published* promijeni, dolazi do ponovnog učitavanja svakog pogleda koji sadrži taj objekt kako bi se nastale promjene primijenile.
- *@State* je omotač svojstva koji omogućuje izmjenu svojstva unutar strukture, što inače nije dozvoljeno jer je struktura vrijednosna vrsta. Korištenjem *@State* premješta se pohrana svojstva iz strukture u zajedničku pohranu kojom upravlja SwiftUI, što znači da SwiftUI može stvoriti i uništiti strukture kad god je potrebno bez gubitka stanja koje je pohranjeno.
- *@FetchRequest* pokreće zahtjev za dohvaćanje podataka iz lokalne memorije (Core Data) za određeni entitet.
- *@NSApplicationDelegateAdaptor* se koristi za stvaranje i registraciju klase koja se koristi kao delegat aplikacije.
- *@SceneStorage* omogućuje spremanje malih količina podataka u lokalnu memoriju koji se kasnije mogu koristiti za vraćanje određenog stanje aplikacije.

3.3. Combine radni okvir

Combine je radni okvir koji je Apple predstavio na WWDC 2019. godine. Combine pruža deklarativni Swift API za obradu vrijednosti tijekom vremena. Vrijednosti predstavljaju vrste asinkronih događaja. Combine deklarira izdavače (eng. *publisher*) da izlažu vrijednosti koje se mogu promijeniti i deklarira pretplatnike (eng. *subscriber*) da primaju vrijednosti od izdavača. Combine se najbolje koristi kada se želi reagirati na različite ulaze (eng. *input*). Korisnička sučelja se vrlo dobro uklapaju u ovaj obrazac, te tako naprimjer Combine operatori omogućuju reagiranje na korisnikovo dinamičko unošenje vrijednosti u *TextField* i ažuriranje pogleda korisničkog sučelja bazirano na korisnikov unos. Combine nije ograničen samo na korisničko sučelje. Bilo koji slijed asinkronih operacija se može uklopiti u Combine posebno kada rezultati svake operacije prelaze na sljedeću operaciju. Combine se može koristiti i za definiranje načina rukovanja pogreškama asinkronih operacija [13].

3.3.1. Izdavač podataka

Izdavač predstavlja davatelja podataka. Prema [13] *Publisher* protokol omogućuje vraćanje podataka pretplatniku kada on zatraži od izdavača. *Just* i *Future* omogućuju stvaranje izdavača iz vrijednosti ili asinkrone funkcije. Većina izdavača odmah vraća podatke kada ih pretplatnik zatraži. Postoje slučajevi u kojima izdavač ima zaseban mehanizam koji mu omogućuje vraćanje podataka nakon pretplate što je omogućeno sa *ConnectablePublisher* protokolom. Izdavač koji implementira ovaj protokol će imati dodatni mehanizam za pokretanje protoka podataka nakon što pretplatnik zatraži zahtjev. Combine nudi niz dodatnih izdavača kao što su *Just, Failure, Fail, Share, Record, Empty, Sequence* i *Multicast* [13].

3.3.2. Pretplatnik na podatke

Kako je navedeno u [13] pretplatnik predstavlja primatelja podataka od izdavača. *Subscriber* protokol omogućuje primanje podataka od raznih Combine operatora. *Assign* i *sink* predstavljaju dva tipa pretplatnika koje nudi Combine. Vrsta pretplatnika koje nudi SwiftUI je *onReceive*. Pretplatnici mogu podržavati otkazivanje (eng. *cancellation*) koje prekida pretplatu i zaustavlja primanje podataka. *Assign* i *sink* implementiraju *Cancellable* protokol. *Assign* primjenjuje vrijednosti proslijeđene od izdavača do objekta definiranog putem, dok *sink* prihvaća zatvoreni izraz koji prima

sve rezultirajuće vrijednosti od izdavača koje omogućuju programeru prekidanje Combine operatora. *Sink* pretplatnik je od velike pomoći prilikom pisanja jediničnih testova (eng. *Unit test*).

3.3.3. Operatori nad podacima

Operatori su naziv za brojne unaprijed izgrađene funkcije koje su uključene u *Publisher* protokol. Prema [14] većinom se koristi niz operatora, jedan za drugim, kako bi se postigli željeni rezultati. Mnogi operatori prihvaćaju zatvoreni izraz od strane programera kako bi se definirala poslovna logika uz pridržavanje životnog ciklusa izdavača i pretplatnika. Operatori se mogu koristiti i za rukovanje pogreškama. Najkorišteniji operatori su:

- *Map* operator koji se koristi za pretvaranje jednog tipa podatka u drugi. Pruža mogućnost manipulacije podacima ili vrstom podataka i najkorišteniji je operator. Ne dopušta izbacivanje pogrešaka i ne pretvara tip pogreške.
- *TryMap* je sličan *Map* operatoru, ali za razliku od njega podržava izbacivanje pogrešaka ako je pretvaranje podataka neuspješno.
- *Scan* prikuplja i mijenja vrijednosti prema logici zatvorenog izraza i vraća međurezultate sa svakom promjenom operatora koje se izvode prije njega.
- *FlatMap* se koristi s asinkronim operacijama koje bi mogle biti neuspješne. *FlatMap* će sve dolazne vrijednosti zamijeniti drugim izdavačem. Kao parametar sadrži zatvoreni izraz koje omogućuje čitanje vrijednosti dolaznih podataka i stvara izdavača koji vraća vrijednost idućem operatoru.
- *Filter* prolazi kroz sve instance izlaznog tipa podatka koje odgovaraju navedenom zatvorenom izrazu odbacujući sve podatke koji se ne podudaraju s uvjetom. Kao parametar prima zatvoreni izraz koje sadrži vrijednost prošlog operatora i vraća Bool vrijednost. Ako je vrijednost istinita ona se šalje idućem operatoru, dok se u suprotnom slučaju vrijednost odbaciva.
- *Retry* je operator koji se koristi kako bi se ponovio zahtjev prethodnom izdavaču u slučaju pogreške.

Commented [JB8]: Premali odlomak

Commented [JB9]: Premali odlomci

Programski kod 3.14. prikazuje primjer korištenja Combine radnog okvira.

```
import UIKit
import Combine

var cancellables: Set<AnyCancellable> = .init()

Just(1)
   .map { value -> String in
      return "Value is \(value)"
   }
   .sink { receivedValue in
      print(receivedValue)
   }
   .store(in: &cancellables)

// Value is 1
```

Programski kod 3.14. Primjer korištenja Combine radnog okvira

3.3.4. Subjekt izdavač podataka

Subjekt je posebna vrsta izdavača koji implementira *Subject* protokol. Protokol zahtjeva od subjekta sadržavanje *send* funkcije koja omogućuje slanje vrijednosti pretplatnicima. Subjekt se koristi i za spajanje više Combine operatora. Subjekt pruža točku agregacije, odnosno subjekt neće signalizirati zahtjev povezanim izdavačima sve dok ne primi barem jednog pretplatnika. Kada primi neku pretplatu, signalizira potražnju podataka od povezanih izdavača. Combine pruža na korištenje dva subjekta: *CurrentValueSubject* i *PassthroughSubject*. Razlika između ova dva subjekta je što *CurrentValueSubject* pamti i zahtjeva početno stanje (vrijednost), dok *PassthroughSubject* ne. Oba subjekta šalju podatke svojim pretplatnicima kada je funkcija *send* pozvana. I *CurrentValueSubject* i *PassthroughSubject* su korisni za stvaranje izdavača za objekte koji implementiraju *ObservableObject* protokol. *ObservableObject* protokol podržava niz deklarativnih komponenti unutar SwiftUI-a. Subjekti su najkorišteniji izdavači podataka u aplikacijama koje sadrže MVVM (Model-View-ViewModel) arhitekturu [13].

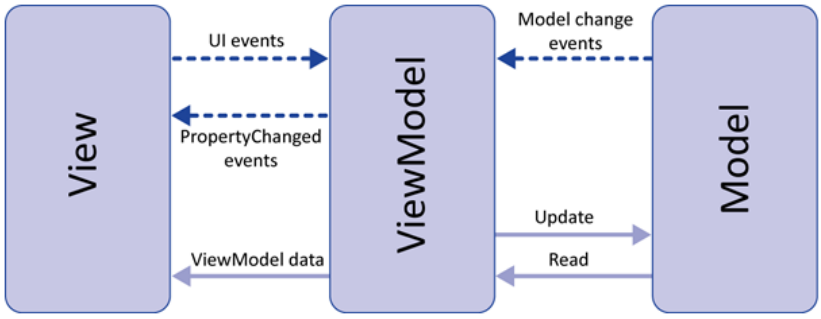
3.4. Model - View - ViewModel arhitektura

MVVM (Model-View-ViewModel) je obrazac koji poboljšava organiziranje koda tako da razdvaja logiku korisničkog sučelja od pozadinske logike. Korištenjem MVVM-a olakšano je jedinično testiranje aplikacije. Uz MVP (Model-View-Presenter) predstavlja najpopularniju arhitekturu za izradu mobilnih aplikacija.

Prema [14] model predstavlja podatke i poslovnu logiku aplikacije. Preporučena strategija implementacije ovog sloja je izlaganje njegovih podataka putem promatrača kako bi se u potpunosti odvojili od ViewModela. Ako dođe do nekih promjena u sloju pogleda (npr. korisnik je pritisnuo dugme), promjene se komuniciraju prema modelu preko ViewModela. Ako dođe do promjene modela, potrebno je ažurirati korisničko sučelje, te model šalje promjene prema ViewModelu.

Uloga pogleda (eng. *View*) u ovom obrascu je promatrati ViewModel kako bi primio podatke s kojima će se ažurirati elementi korisničkog sučelja. Pogled ne može direktno komunicirati s modelom. Komunikacija se izvršava preko ViewModela.

ViewModel sadrži poslovnu logiku, komunicira s modelom i priprema podatke koji će biti prikazani u pogledu. Predstavlja komunikatora između modela i pogleda. Na slici 3.2. je prikazan primjer MVVM arhitekture.



Sl. 3.2. Komponente i komunikacija MVVM-a prema [14]

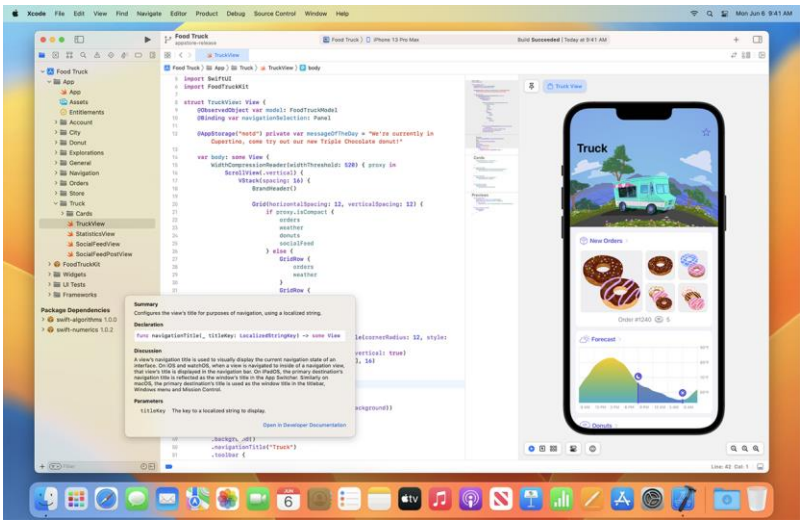
Commented [JB10]: Ako nije vaša slika navesti izvor tako da broj litarature gdje je naveden link stavite u naziv slike

Primjer korištenja MVVM-a je kada korisnik izvrši ažuriranje korisničkog sučelja (npr. ažuriranje e-mail adrese putem *TextFielda*), te se vrijednost u *TextFieldu* šalje *ViewModelu*. *ViewModel* zatim ažurira model koji treba promijeniti adresu e-maila korisničkog računa. Model zatim šalje ažuriranja natrag *ViewModelu* i obavještava o uspješnosti ažuriranja. *ViewModel* nova ažuriranja šalje natrag pogledu koji ažurira korisničko sučelje. Pogled zatim može prikazati korisniku informaciju o tome je li promjena e-mail adrese bila uspješna ili ne [15].

Commented [JB11]: Praznina ispod

3.5. Xcode razvojno okruženje

Xcode je Appleovo integrirano razvojno okruženje (eng. *Integrated development environment*) za macOS, koje se koristi za razvoj softvera za macOS, iOS, iPadOS, watchOS i tvOS. Objavljen je 2003. godine i dostupan je besplatno putem Mac App Storea. Xcode podržava C, C++, Objective-C, Java, Python, Ruby i Swift programske jezike. Koristeći iOS SDK (*Software development kit*), watchOS SDK, Xcode se koristi za kompajliranje i otklanjanje pogrešaka u Apple aplikacijama. Xcode integrira ugrađenu podršku za upravljanje izvornim kodom korištenjem Gitovog sustava kontrole verzija dopuštajući korisniku stvaranje i kloniranje Git repozitorija, te korištenje Git operacija koje bi se tradicionalno koristile korištenjem Git naredbenog retka. Slika 3.3. prikazuje Xcode korisničko sučelje [15].



Sl. 3.3. Xcode korisničko sučelje prema [15]

4. RAZVOJ KORISNIČKE APLIKACIJE

Aplikacija se sastoji od četiri zaslona: *HomeScreen*, *MapScreen*, *LatestNewsScreen* i *CovidScannerScreen*. Početni pogled koji je prikazan prilikom pokretanja aplikacije je *HomeView* koji se nalazi unutar *HomeContainerView* pogleda.

HomeScreen zaslon prikazuje slučajeve koronavirusa ovisno o tome je li izabran *worldwide* ili *country* slučaj. Ako je slučaj *worldwide*, prikazuju se podaci od tri države s najviše slučajeva koronavirusa. *Country* slučaj prikazuje podatke za pojedinu državu. Na zaslonu su prikazani i podaci poput stope smrtnosti, vjerojatnosti oporavka itd.

Ako je izabran *worldwide* slučaj, *MapScreen* prikazuje geografsku kartu svijeta i oznake na tri države s najviše slučajeva koronavirusa. Ako se pritisne jedna od oznaka, zaslon prikazuje podatke o koronavirusu za tu državu. *Country* slučaj na *MapScreenu* prikazuje geografsku kartu države koja je izabrana i podatke o koronavirusu za tu državu. Ako se pritisne oznaka na karti prikazuju se osnovni podaci o državi. *Country* slučaj sadrži i gumb čijim se pritiskom otvara nativna *Maps* aplikacija, te ona prikazuje udaljenost korisnika od položaja oznake na mapi.

LatestNewsScreen prikazuje listu najnovijih vijesti vezanih za koronavirus. Ako se pritisne na neku vijest, otvara se *WebView* pogled koji prikazuje potpuni članak vezan za tu vijest. Pogled je prikazan unutar nativne aplikacije internetskog preglednika.

CovidScannerScreen se sastoji od kamere s kojom je omogućeno skeniranje QR koda COVID digitalne potvrde. Ako je skeniranje uspješno, prikazuju se podaci o vlasniku certifikata. U slučaju neuspješnog skeniranja prikazuje se poruka, te je omogućen odabir ponovnog skeniranja. Nakon uspješnog skeniranja COVID digitalna potvrda je spremljena u lokalnu memoriju mobilnog uređaja.

U početnom pogledu aplikacije kreiran je *TabView* s četiri slučaja koji predstavljaju zaslone aplikacije. U pogledu je kreirana *UseCaseSelection* enumeracija s *worldwide* i *country* slučajevima. Početna vrijednost varijable *useCase*, koja je tipa *UseCaseSelection*, je *worldwide*. Programski kod 4.1 prikazuje stvaranje enumeracija i strukture od kojih se sastoji početni pogled aplikacije. Struktura *HomeContainerView* implementira *View* protokol.

```
enum TabIcon: String {
   case home = "homeTabIcon"
   case latestNews = "latestNewsTabIcon"
   case statistics = "statisticsTabIcon"
   case pageComingSoon = "pageComingSoonTabIcon"
}

enum UseCaseSelection: Equatable {
   case worldwide
   case country(String)
}

struct HomeContainerView: View {

   @State var useCase: UseCaseSelection = .worldwide

   var body: some View {
      TabView {
         HomeView(viewModel: HomeViewModel(repository: StatisticsRepositoryImpl(),
useCase: useCase), useCase: $useCase)
            .tabItem {
               renderAsImage(.home)
            }

         MapStatisticsView(viewModel: MapViewModel(repository:
StatisticsRepositoryImpl(), useCase: useCase), useCase: $useCase)
            .tabItem {
               renderAsImage(.statistics)
            }

         LatestNewsView(viewModel: LatestNewsViewModel(repository:
LatestNewsRepositoryImpl()))
            .tabItem {
               renderAsImage(.latestNews)
            }

         CovidCertificatesView(viewModel: CovidCertificatesViewModel(repository:
CovidScannerRepositoryImpl()))
            .tabItem {
               renderAsImage(.pageComingSoon)
            }
      }
      .accentColor(.red)
   }
```

Programski kod 4.1. Struktura *HomeContainerView* i enumeracije *TabIcon* i *UseCaseSelection*

4.1. Rukovanje logikom mrežnog sloja

Klasa *RestManager* sadrži generičku funkciju *fetchData* koja kao parametar prima *URL*, a kao povratnu vrijednost vraća *AnyPublisher<T, ErrorType>*. Ova funkcija koristi se za sve API pozive koji su korišteni u aplikaciji. Funkcija izvršava dekodiranje vrijednosti dohvaćene s API-ja na temelju kreiranih JSON modela. Ako je dekodiranje uspješno u izlazni podatak *AnyPublishera* sprema se dekodirana vrijednost, dok se u slučaju neuspješnog dekodiranja, na temelju pogreške koje je stvorilo neuspješno dekodiranje, u podatak greške sprema vrijednost slučaja *ErrorType* enumeracije. *ErrorType* sadrži *general*, *empty* i *noInternetConnection* slučajeve. Funkcija je prikazana programskim kodom 4.2.

```
static func fetchData<T: Decodable>(url: URL) -> AnyPublisher<Result<T, ErrorType>,
Never> {
    URLSession
      .shared
      .dataTaskPublisher(for: url)
      .mapError { urlError -> ErrorType in
        switch urlError.code {
        case .notConnectedToInternet, .networkConnectionLost, .timedOut:
          return ErrorType.noInternetConnection
        case .cannotDecodeRawData, .cannotDecodeContentData:
          return ErrorType.empty
        default:
          return ErrorType.general
        }
      }
      .map { $0.data }
      .decode(type: T.self, decoder: JSONDecoder())
      .map { result -> Result<T, ErrorType> in
        if let array = result as? Array<Any>, array.isEmpty {
          return Result.failure(ErrorType.empty)
        }
        return Result.success(result)
      }
      .catch { Just<Result<T, ErrorType>>(.failure($0 as? ErrorType ?? .general))}
      .eraseToAnyPublisher()
  }
```

Programski kod 4.2. Funkcija *fetchData*

RestEndpoints enumeracija sadrži slučajeve *countries*, *countriesStats*, *worldwideStats* i *latestNews*, *vaccoid* i *restCountries*. Također su deklarirane i statičke varijable koje sadrže vrijednost linkova za određeni slučaj. Funkcija *endpoint* koristi naredbu *switch*, te ovisno o izabranom slučaju vraća *URL*. Enumeracija se koristi s *RestManager* klasom unutar repozitorij klasa za određeni zaslon. Programski kod 4.3 prikazuje *RestEnpoints* enumeraciju.

```
enum RestEndpoints {
    case countries
    case countryStats(name: String)
    case worldwideStats
    case latestNews(Int)
    case vaccoid
    case restCountries

    static let covidHost = "api.covid19api.com"
    static let newsHost = "api.mediastack.com/v1/news?"
    static let newsAccessKey = "access_key=1fbdf350fa58d99830a7c17f6cde"
    static let vaccoidHost = "https://vaccovid-coronavirus-vaccine-and-treatment-
tracker.p.rapidapi.com/api/
    static let restCountriesHost = "https://restcountries.com/v3.1/all"

    static var ENDPOINT_COVID: String {
        return "https://" + covidHost
    }
    static var ENDPOINT_NEWS: String {
        return "http://" + newsHost + newsAccessKey
    }
    public func endpoint() -> URL {
        switch self {
        case .countries:
            return URL(string: RestEndpoints.ENDPOINT_COVID + "/countries")!
        case .countryStats(let name):
            return URL(string: RestEndpoints.ENDPOINT_COVID + "/dayone/country/" +
name)!
        case .worldwideStats:
            return URL(string: RestEndpoints.ENDPOINT_COVID + "/summary")!
        case .latestNews(let offset):
            return URL(string: RestEndpoints.ENDPOINT_NEWS +
"&keywords=coronavirus&offset=\(offset)&languages=en&sort=published_desc&limit=25
")!
        }
    }
```

Programski kod 4.3. Enumeracija *RestEnpoints*

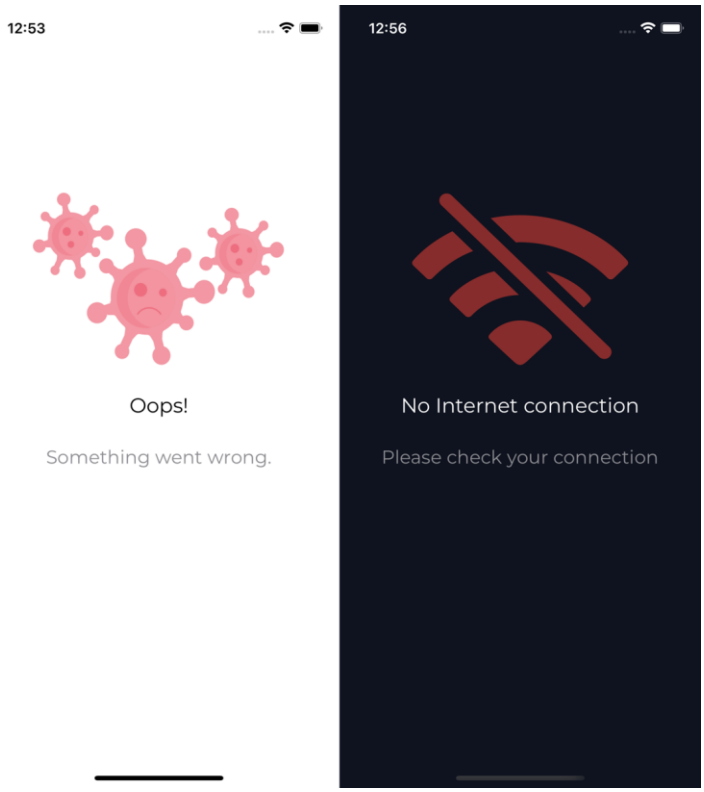
4.2. Pogledi pogrešaka i učitavanja

Prilikom dekodiranja podataka koji su preuzeti s interneta može doći do pogreške dekodiranja. Korisniku je zatim potrebno prikazati pogrešku na zaslonu. *General* predstavlja pogrešku koja je nastala prilikom dekodiranja ili zbog nekog drugog razloga prilikom pripreme podataka koji će biti prikazani korisniku. *Empty* je pogreška koja se prikazuje kada je dekodiranje uspješno, ali niz u kojem se nalazi rezultat je prazan. *NoInternetConnection* predstavlja pogrešku kada uređaj nema internet povezanost, te zbog toga nije moguće preuzeti podatke s interneta. *General* i *NoInternetConnection* pogreške su prikazane u *ErrorView* pogledu. Programskim kodom 4.4 prikazan je *ErrorView* pogled.

```
VStack(spacing: 30) {
        switch errorType {
        case .general:
            Image("generalError")
                .resizable()
                .foregroundColor(Color.red.opacity(0.5))
                .frame(width: width * 0.7, height: width / 2.1)

        case .noInternetConnection:
            Image(systemName: "wifi.slash")
                .resizable()
                .foregroundColor(Color.red.opacity(0.5))
                .frame(width: width * 0.6, height: width / 2.1)
        }
        Text(verbatim: errorType == .noInternetConnection ? .noInternetErrorTitle :
.generalErrorTitle)
            .commonFont(.regular, style: .title2)

        Text(verbatim: errorType == .noInternetConnection ? .noInternetErrorDescription
: .generalErrorDescription)
            .commonFont(.regular, style: .title3)
            .foregroundColor(Color(UIColor.systemGray))

        if let errorActionCallback = buttonAction {
            Button(errorType == .noInternetConnection ? String.noInternetErrorButtonTitle
: String.generalErrorButtonTitle) {
                errorActionCallback()
            }
            .coronaVirusAppButtonStyle()
        }
```

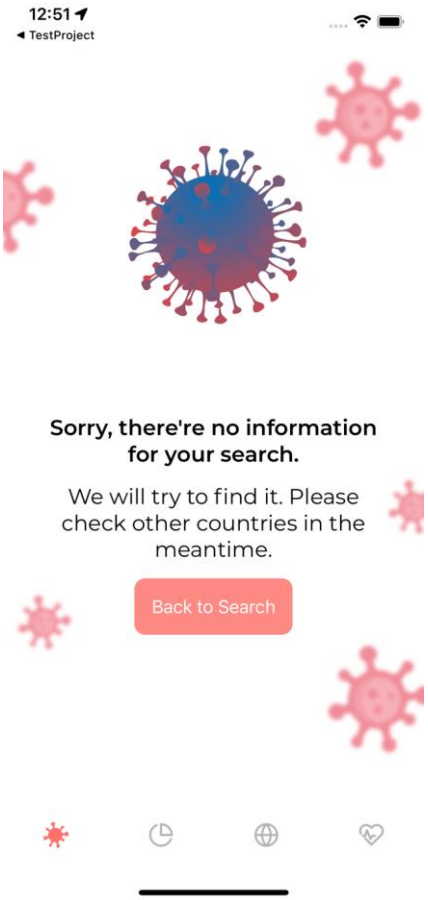
Programski kod 4.4. Pogled *ErrorView*

Slika 4.1., ovisno o slučaju pogreške, prikazuje izgled *ErrorView* pogleda na mobilnom uređaju. Pritiskom na gumb poziva se repozitorij funkcija za ponovno dohvaćanje podataka s interneta ovisno o zaslonu na kojem je došlo do pogreške.



Sl. 4.1. Izgled *ErrorView* pogleda ovisno o slučaju pogreške

EmptyStateView pogled prikazuje se kada je niz u koji se sprema dekodirana vrijednost prazan. Pogled se prikazuje kada izborom nove države u *CountrySelectionView* pogledu država nema dovoljno podataka o koronavirus slučajevima. Pogled sadrži animaciju koja je prikazana koristeći *Lottie* biblioteku. Pritiskom na gumb otvara se *CountrySelectionScreen* pogled kako bi se mogao odabrati novi *UseCaseSelection* slučaj. Pogled sadrži parametar *isShowingCountrySelection* tipa *Binding* čija vrijednost predstavlja prikazivanje *CountrySelectionScreen* pogleda, odnosno vrijednost koja se mijenja pritiskom na dugme. Izgled *EmptyStateView* pogleda prikazan je na slici 4.2.



Sl. 4.2. Izgled *EmptyStateView* pogleda

Tijekom preuzimanja podataka s interneta prikazuje se *LoadingView* pogled sve dok podaci nisu uspješno preuzeti. *ViewModeli* klasa zaslužni su za rukovanje prikazivanja pogleda tijekom poziva repozitorij funkcija za preuzimanje podataka s interneta. Nakon *LoadingView* pogleda prikazuje se pogled s podacima ili pogled s pogreškom, ovisno o uspješnosti dekodiranja. Pogled sadrži animaciju koja je prikazana koristeći *Lottie* biblioteku.

4.3. Zaslon s podacima o koronavirusu

Nakon što je aplikacija otvorena prikazuje se *HomeView* pogled. Pogled prikazuje podatke o slučajevima koronavirusa ovisno o *UseCaseSelection* varijabli koja se kao parametar predaje pogledu. Pogledu se kao parametar predaje i *HomeViewModel*. *HomeViewModel* priprema podatke koji će biti prikazani na pogledu. Kao parametar prima *StatisticsRepository* koji dohvaća podatke s interneta i omogućuje *HomeViewModelu* rad s tim podacima. Kao parametar prima i *UseCaseSelection* varijablu čija vrijednost predstavlja hoće li repozitorij dohvatiti podatke za pojedinu državu (*country* slučaj) ili podatke za tri države s najviše slučajeva koronavirusa (*worldwide* slučaj). Nakon što su podaci dohvaćeni oni se predaju *HomeDomainItemu* kao parametar koji će ekstraktirati potrebne podatke. *StatisticsRepository* sadrži funkcije *getCountryData* i *getWorldwideData*. Funkcija *getCountryData* kao parametar prima ime države za koju će se dohvatiti podaci s interneta. Unutar funkcije, koristeći *RestManager*, podaci se dekodiraju i ovisno o uspješnosti dekodiranja funkcija vraća izdavača sa sortiranim podacima o slučajevima koronavirusa na temelju najranijeg datuma ili pogrešku koja je uzrokovala neuspješnost dekodiranja. Funkcija *getWorldwideData* vraća izdavača s podacima o tri države s najviše slučajeva koronavirusa ili pogrešku koja je dovela do neuspješnosti dekodiranja. Programski kod 4.5 prikazuje *getCountryData* funkciju.

```
func getCountryData(for name: String) ->
AnyPublisher<Result<[CountryDayOneResponseItem], ErrorType>, Never> {
      let url = RestEndpoints.countryStats(name: name).endpoint()
      let publisher: AnyPublisher<Result<[CountryDayOneResponseItem], ErrorType>,
Never> = RestManager.fetchData(url: url)

      return publisher
         .map { response -> Result<[CountryDayOneResponseItem], ErrorType> in
            switch response {
            case .success(let countries):
               return Result.success(countries.sorted(by: { $0.date > $1.date }))
            case .failure(let error):
               return Result.failure(error)
            }
         }
         .eraseToAnyPublisher()
   }
```

Programski kod 4.5. Funkcija *getCountryData*

StatisticsRepository sadrži i funkcije *getVaccoidCovidData* i *getCountriesInformations* kako bi se dohvatili dodatni podaci s interneta potrebni za prikaz detaljnijih informacija o slučajevima koronavirusa i detaljima o državama. *HomeViewModel* kao parametar prima objekt *StatiscticsRepositorya* i *useCase* koji je tipa *UseCaseSelection*. Parametar *useCase* se predaje subjekt izdavaču podataka koji je tipa *CurrentValueSubject* kako bi se dohvatili potrebni podaci za *worldwide* ili *country* slučaj. Svojstva i konstruktor *HomeViewModela* prikazani su programskim kodom 4.6.

```
    private let repository: StatisticsRepository

    @Published var useCase: UseCaseSelection
    @Published var error: ErrorType?
    @Published var data: HomeDomainItem?

    @Published var loader = true
    @Published var isShowingCountrySelection = false
    @Published var date = Date()

    private let useCaseSelectionSubject: CurrentValueSubject<UseCaseSelection, Never>

    private var countryData: [CountryDayOneResponseItem] = []
    private var coronaData: [VaccoidCoronaResponseItem] = []

    private var cancellables: Set<AnyCancellable> = .init()

    init(repository: StatisticsRepository, useCase: UseCaseSelection){
        self.repository = repository
        self.useCase = useCase
        self.useCaseSelectionSubject = CurrentValueSubject<UseCaseSelection,
Never>.init(useCase)
        initPipelines()
    }
```

Programski kod 4.6. Svojstva i konstruktor *HomeViewModela*

HomeViewModel sadrži funkciju *initPipelines* u kojoj se poziva repozitorij funkcija za dohvaćanje podataka, te se ti podaci predaju *HomeDomainItemu*. Unutar funkcije se nalazi i varijabla *date* koja predstavlja izdavača podataka, te se njenom promjenom poziva funkcija *updateDate* koja ažurira podatke ovisno o datumu kojeg korisnik izabere na *HomeView* pogledu. Programski kod 4.7 prikazuje funkciju *initPipelines*.

```
private func initPipelines(){
    $date
      .receive(on: RunLoop.main)
      .sink { [weak self ] _ in
        guard let self = self else {return}
        self.updateDate()
      }
      .store(in: &cancellables)

    useCaseSelectionSubject
      .flatMap { [weak self] value -> AnyPublisher<Result<HomeDomainItem,
ErrorType>, Never> in
        guard let self = self else {
          return Just(Result.failure(ErrorType.general)).eraseToAnyPublisher()
        }
        self.loader = true
        self.error = nil
        switch value {
        case let .country(name):
          return self.countryPipeline(name: name)

        case .worldwide:
          return self.worldwidePipeline()
        }
      }
      .receive(on: RunLoop.main)
      .sink { [weak self] response in
        guard let self = self else { return }
        switch response {
        case .success(let item):
          self.data = item
          self.loader = false
          self.error = nil

        case .failure(let error):
          self.loader = false
          self.error = error
        }
      }
      .store(in: &cancellables)
```

Programski kod 4.7. Funkcija *initPipelines*

Varijabla *useCase*, koja je tipa *UseCaseSelection*, omotana je s *Published* omotačem. Na varijablu se poziva *flatMap* operator koji vraća izdavača s vrijednošću *HomeDomainItem* i *ErrorType* tipom pogreške. Unutar operatora se varijabla *loader* postavlja na *true* kako bi se na *HomeView* pogledu mogao biti prikazivati *LoadingView* pogled sve dok podaci nisu preuzeti. Na temelju *useCase* vrijednosti, koristeći *switch*, poziva se *countryPipeline* ili *worldwidePipeline* funkcija. Funkcije pozivaju repozitorij funkcije ovisno o vrijednosti *useCasea*, te povratne vrijednosti funkcije predaju *HomeDomainItemu* kao parametar. Funkcije su označene s *private* jer se koriste samo unutar *HomeViewModel* klase. Unutar funkcije se kreiraju dvije varijable kojima se pridružuju vrijednosti izdavača podataka dobivenih pozivanjem repozitorij funkcija *getVaccoidCovidData* i *getWorlwideData*. Koristeći *Pubishers.Zip* izdavači se pridružuju jedan drugome, te se poziva operator *tryMap* koji iz dobivenog rezultata koristeći *switch* grananje u *success* slučaju sprema vrijednosti iz pojedinog izdavača podataka u novu varijablu koja će se proslijediti *HomeDomanItemu* kao parametar. Razlika između *worldwidePipeline* i *countryPipeline* funkcija je što *countryPipeline* kao parametar prima String objekt koji predstavlja ime države za koju se dohvaćaju podaci, te se poziva *getCountryData* repozitorij funkcija kojoj se kao parametar predaje ime države. Rezultati dobiveni iz *success* slučaja *switch* grananja se spremaju u svojstva klase *HomeViewModel* *countryData* i *coronaData* koje se koriste kako bi se mogli prikazati određeni podaci vezani za određeni datum koji korisnik izabere na *HomeView* pogledu. Nakon što je određena funkcija vratila vrijednost, poziva se *receive* operator kako bi se promjene na korisničkom sučelju mogle izvršiti na glavnoj niti aplikacije. Glavna nit aplikacije (*eng. MainThread*) je nit na kojoj je potrebno izvršavati operacije koje su vezane samo za promjene izgleda korisničkog sučelja kako se ne bi usporio rad aplikacije. Učestalo korištenje glavne niti aplikacije može dovesti do prekida rada aplikacije. *Sink* operator, u slučaju da je povratna vrijednost pogreška, postavlja vrijednost pogreške (koja je *ErrorType*) u varijablu *error* kako bi se određena pogreška mogla prikazati na pogledu, dok u suprotnom slučaju postavlja vrijednost u *data* varijablu koja je tipa *HomeDomainItem*. U oba slučaja se varijabla *loader* postavlja na *false* kako bi se na pogledu mogli prikazati podaci ili pogreška. Programski kod 4.8 prikazuje funkciju *worldwidePipeline*.

```
private func worldwidePipeline() -> AnyPublisher<Result<HomeDomainItem,
ErrorType>, Never> {
let firstPublisher = repository.getVaccoidCovidData()
let secondPublisher = repository.getWorldwideData()

return Publishers.Zip(firstPublisher, secondPublisher)
    .tryMap { data -> Result<HomeDomainItem, ErrorType> in

        var worldwideResponseItem: WorldwideResponseItem
        var vaccoidData: [VaccoidCoronaResponseItem]

        switch data.1 {
                case .success(let worldwideResponse):
                        worldwideResponseItem = worldwideResponse
                case .failure(let error):
                        return Result.failure(error)
        }

        switch data.0 {
                case .success(let vaccoid):
                        vaccoidData = vaccoid
                case .failure(let error):
                        print(error.localizedDescription)
                return Result.failure(error)
        }

        do {
        return try Result.success(HomeDomainItem(response: worldwideResponseItem,
        vaccoidCoronaData: vaccoidData))
        }
        catch let error {
                print(error)
        return Result.failure(error.asErrorType)
        }
}
.mapError { _ in ErrorType.general}
.catch { Just<Result<HomeDomainItem, ErrorType>>(.failure($0)) }
.eraseToAnyPublisher()
}
```

Programski kod 4.8. Funkcija *worldwidePipeline*

Klasa *HomeDomainItem* nasljeđuje klasu *StatisticsDomainItem* i njena svojstva koja će sadržavati podatke potrebne za prikaz na zaslonu. Programskim kodom 4.9 prikazana je klasa *StatisticsDomainItem.*

```
class StatisticsDomainItem {

    var title: String?
    var flag: String?
    var confirmedCases: StatisticsItem?
    var activeCases: StatisticsItem?
    var recoveredCases: StatisticsItem?
    var deathCases: StatisticsItem?
    var vaccoidCorona: [VaccoidCoronaResponseItem]?
    var fatalityRate: Double?
    var recoveryPossibility: Double?
    var deathsPerMillion: Int?
    var seriousCritical: Int?
    var totalCasesPerMillion: Int?

    init(
        title: String? = nil,
        confirmedCases: StatisticsItem? = nil,
        activeCases: StatisticsItem? = nil,
        recoveredCases: StatisticsItem? = nil,
        deathCases: StatisticsItem? = nil,
        vaccoidCorona: [VaccoidCoronaResponseItem]? = nil
    ){
        self.title = title
        self.confirmedCases = confirmedCases
        self.activeCases = activeCases
        self.recoveredCases = recoveredCases
        self.deathCases = deathCases
        self.vaccoidCorona = vaccoidCorona
    }
}
```

Programski kod 4.9. Klasa *StatisticsDomainItem*

HomeDomainItem sadrži dva parametarska konstruktora koja mogu izbacivati pogreške. Prvi konstruktor kao parametar prima niz vrijednosti dohvaćenih za pojedinu državu i niz podataka tipa *VaccoidCoronaResponseItem*. Ako niz ima više od dvije vrijednosti pozivaju su privatne funkcije koje će pripremiti podatke za njihovo prezentiranje na pogledu. Ako niz nema više od dvije vrijednosti

konstruktor izbacuje *ErrorType* pogrešku. Drugi konstruktor prima *WordwideResponseItem* parametar koji predstavlja objekt koji sadrži podatke vezane za tri države s najviše slučajeva koronavirusa i niz podataka tipa *VaccoidCoronaResponseItem*. Drugi konstruktor je isto označen s *throws*, što znači da konstruktor može izbaciti pogrešku kojom se upravlja unutar *HomeViewModela*. Programski kod 4.10 prikazuje konstruktore klase *HomeDomainItem*.

```
init(response: [CountryDayOneResponseItem], vaccoidCoronaData:
[VaccoidCoronaResponseItem]) throws {
        super.init()
        if response.count > 2 {
            vaccoidCorona = vaccoidCoronaData
            title = response[0].name
            setCountryData(coronaData: vaccoidCoronaData, countryData: response)
            updateDate(dateString: response[0].date)
        } else {
            throw ErrorType.general
        }
    }

    init(response: WorldwideResponseItem, vaccoidCoronaData:
[VaccoidCoronaResponseItem]) throws {
        super.init()
        if !vaccoidCoronaData.isEmpty {
            title = "Worldwide"
            vaccoidCorona = vaccoidCoronaData
            setWorldwideStats(items: vaccoidCoronaData)
            updateDate(dateString: response.date)
            setWorldwideData()
        } else {
            throw ErrorType.general
        }
    }
```

Programski kod 4.10. Konstruktori *HomeDomainItema*

Funkcija *setCountryData* kao parametar prima objekte koji su predani *HomeDomainItem* konstruktoru. Unutar funkcije se kreiraju dvije *for* petlje s kojima se prolazi kroz oba niza podataka dobivenih iz parametara. Ako svojstvo *name* iz objekta niza podataka tipa *CountryDayOneResponseItem* sadrži istu vrijednost kao i svojstvo *country* iz objekta niza podataka tipa *VaccoidCoronaResponseItem*, svojstvima *HomeDomainItem* klase se pridružuju vrijednosti

dobivene iz niza podataka tipa *VaccoidCoronaResponseItem*, te se koristi *break* kako bi se petlje zaustavile. Programskim kodom 4.11 prikazana je funkcija *setCountryData*.

```
private func setCountryData(coronaData: [VaccoidCoronaResponseItem], countryData:
[CountryDayOneResponseItem]) {
      for item in coronaData {
         for country in countryData {
            if item.country == country.name {
               confirmedCases = StatisticsItem(type: .confirmed,
                                    value: item.totalCases,
                                    delta: item.newCases)

               deathCases = StatisticsItem(type: .deaths,value: item.totalDeaths, delta:
item.newDeaths)

               recoveredCases = StatisticsItem(type: .recovered,
                                    value: Int(item.totalRecovered) ?? 0,
                                    delta: item.newRecovered)

               activeCases = StatisticsItem(type: .active,
                                 value: item.activeCases,
                                 delta: 0)

               fatalityRate = item.caseFatalityRate
               recoveryPossibility = item.recoveryProporation
               deathsPerMillion = Int(item.deaths1MPop)
               seriousCritical = item.seriousCritical
               totalCasesPerMillion = Int(item.totCases1MPop)
               break
            }
         }
      }
```

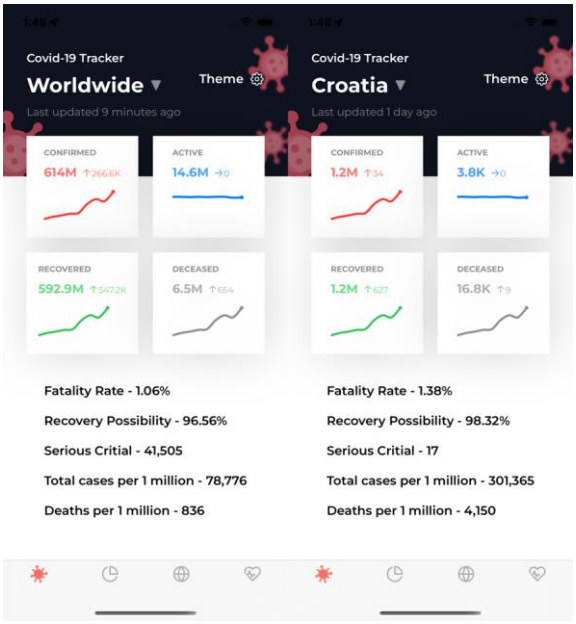
Programski kod 4.11. Funkcija *setCountryData*

Funkcija *setWorldwideListStats* je funkcija koja kreira podatke za listu koja se prikazuje u *HomeView* pogledu. U slučaju da je *useCase country* umjesto liste je prikazan *DatePicker*. Lista u slučaju *worldwide* prikazuje ukupne brojke svih slučajeva za tri države s najviše ukupnih slučajeva. Funkcija kao parametar prima podatke dobivene iz repozitorija u obliku niza. *For* petljom se prolazi kroz niz, te se koristeći klasu *DataListItem* stvaraju objekti koji se dodaju u svojstvo klase *listStats*. Programski kod 4.12 prikazuje funkciju *setWorldwideListStats*.

```
private func setWorldwideStats(items: [VaccoidCoronaResponseItem]) {
    var newStats = [DataListItem]()

    for (index,item) in items.enumerated() {
      if (index > 5) {
        break
      }

      if (index != 0) && (index != 1) && (index != 2){
        let newItem = DataListItem(
          title: item.country,
          confirmed: item.totalCases.abbreviate(),
          recovered: Int(item.totalRecovered)?.abbreviate() ?? "",
          deaths: item.totalDeaths.abbreviate(),
          active: item.activeCases.abbreviate()
        )

        newStats.append(newItem)
      }
    }

    listStats = newStats
  }
```

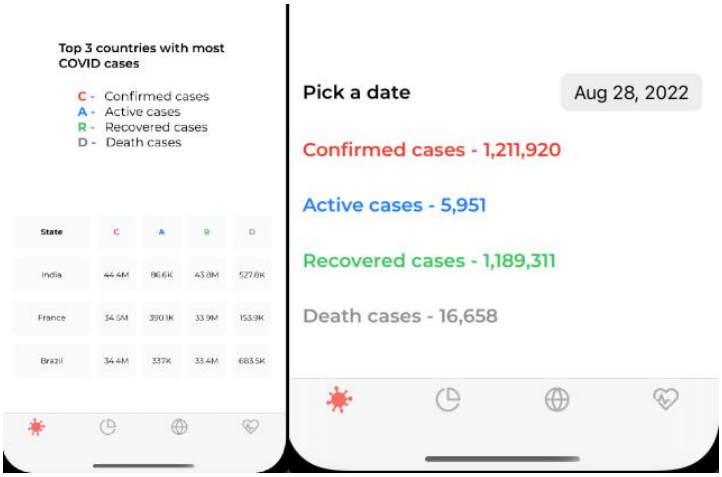
Programski kod 4.12. Funkcija *setWorldwideListStats*

HomeView pogled u oba *useCase* slučaja prikazuje četiri kvadratića u kojima se nalaze informacije o slučajevima koronavirusa i informacije kao što su mogućnost oporavka, broj kritičnih slučajeva, stopa smrtnosti itd. Kvadratići sadrže ukupan broj slučajeva i broj koji prikazuje promjenu slučajeva u odnosu na prošli dan. Ako nije došlo do promjene graf će prikazivati ravnu liniju, dok u slučaju promjene ukupnog broja slučajeva graf raste. Slika 4.3. prikazuje izgled *HomeView* pogleda u oba slučaja kada je *useCase* globalna varijabla *worldwide* i *country* slučaj. U *worldwide* slučaju *HomeView* pogled još sadrži informacije o slučajevima koronavirusa u tri države s najviše slučajeva, dok *country* slučaj sadrži *DatePicker* pomoću kojeg korisnik može izabrati određeni datum, te su zatim prikazane informacije o koronavirus slučajevima za taj datum. Slika 4.4. prikazuje dodatne elemente korisničkog sučelja za oba slučaja u *HomeView* pogledu.



Sl. 4.3. Izgled *HomeView* pogleda

.



Sl. 4.4. Dodatni elementi korisničkog sučelja na *HomeView* pogledu

Unutar zaglavlja pogleda se nalazi tekst koji predstavlja naslov aplikacije (*Covid-19 Tracker*), gumb čiji naziv ovisi o vrijednosti globalne varijable *useCase*, tekst koji prikazuje kada su zadnji put ažurirani podaci na internetu koji se prikazuju na pogledu i gumb za odabir svijetlog ili tamnog načina rada aplikacije. *HomeView* pogled prikazan je programskim kodom 4.13.

```
ZStack(alignment: .top){
        VStack(spacing: 0) {
                ZStack {
                    Rectangle()
                        .fill(Color.appBackground)
                        .frame(width: geo.size.width, height: geo.size.height * 0.33)
                        .background(Color.appBackground)
                        .environment(\.colorScheme, .dark)

                    virusImage(blur: 0)
                        .frame(width: (geo.size.width * 0.3), height: (geo.size.width * 0.3))
                        .offset(x: -(geo.size.width * 0.45), y: geo.size.height * 0.12)
                    virusImage(blur: 1)
                        .frame(width: (geo.size.width * 0.2), height: (geo.size.width * 0.2))
                        .offset(x: geo.size.width * 0.46, y: -(geo.size.height * 0.05))
                    virusImage(blur: 1.5)
                        .frame(width: (geo.size.width * 0.15), height: (geo.size.width *
0.15))
                        .offset(x: geo.size.width * 0.45, y: geo.size.height * 0.1)
                    }
                    Rectangle()
                       .fill(Color.appBackground)
                    }
                VStack {
                    header(data: data)
                       .padding(.top)
                    dashboards(data: data)
                    coronaInformations(data: data)
                       .padding(.bottom)
                    list(data: data)
                 }
                    .padding()
            }
```

Programski kod 4.13. *HomeView* pogled

Lista prikazuje tri države s najviše slučajeva koronavirusa kada je *useCase* vrijednost *worldwide*, odnosno prikazuje broj slučajeva za određeni datum kada *useCase* sadrži vrijednost *country*. Pogled sadrži funkciju *header* unutar koje su elementi raspoređeni vodoravno. Parametar funkcije je *data* koji je tipa *HomeDomainItem*. Elementi korišteni u funkciji su tekst koji predstavlja naslov aplikacije, gumb čijim pritiskom se otvara *CountrySelectionView* pogled i tekst koji predstavlja datum zadnjeg ažuriranja podataka na internetu. Funkcija *dashboards* koristi se za prikaz četiri kvadratića s vrijednostima. Elementi su raspoređeni okomito, te funkcija prima konstantu *data*, tipa *HomeDomainItem* kao parametar. *List* funkcija koristi se za prikaz liste, odnosno za prikaz *DatePicker* pogleda u *country useCase* slučaju. Elementi su raspoređeni unutar *LazyVStacka* kako bi se podaci renderirali kada ih je potrebno prikazati. Lista u *worldwide* slučaju je kreirana koristeći *ForEach* petlju, te svaki element liste predstavlja *HomeListItemView* objekt kojemu se kao parametri predaju vrijednosti dobiveni iz parametra *list.*

Pogled sadrži funkciju *errorViewBuilder* koja se koristi za prikaz pogreške ako svojstvo *error* unutar klase *HomeViewModel* sadrži vrijednost. Funkcija kao parametar prima tu vrijednost, te koristeći *switch* grananje prikazuje *ErrorView* pogled ako su pogreške *general* ili *noInternetConnection*, te *EmptyStateView* pogled ako je pogreška *empty*. Unutar grananja, u slučaju *empty*, pogledu *EmptyStateView* se predaje *isShowingCountrySelection* varijabla kako bi se omogućilo rukovanje ponovnog izbora države. Programski kod 4.14 prikazuje *errorViewBuilder* funkciju.

```
@ViewBuilder
   private func errorViewBuilder(_ error: ErrorType) -> some View {
      switch error {
      case .general:
         ErrorView(.general) {
            viewModel.errorActionCallback()
         }
      case .noInternetConnection:
         ErrorView(.noInternetConnection) {
            viewModel.errorActionCallback()
         }
      case .empty:
         EmptyStateView(isShowingCountrySelection:
$viewModel.isShowingCountrySelection)
      }
```

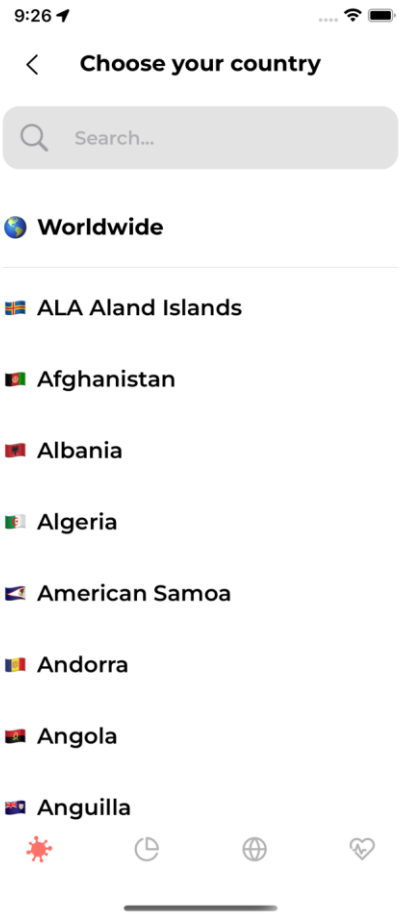
Programski kod 4.14. Funkcija *errorViewBuilder*

Mijenjanje *useCase* varijable omogućeno je u *CountrySelectionView* pogledu. Pogled prikazuje listu država dobivenih iz *CountrySelectionRepository* repozitorija. *CountrySelectionViewModel* klasa sadrži funkciju *initPipelines* koja dohvaća vrijednosti dobivenih iz repozitorija, te zatim sortira vrijednosti kako bi bili abecedno poredani i sprema sortirane vrijednosti u *countries* svojstvo klase. Programski kod 4.15 prikazuje funkciju *initPipelines.*

```
private func initPipelines() {
    countriesSubject
      .flatMap { [weak self] _ -> AnyPublisher<Result<[CountryDetails], ErrorType>,
Never> in
        guard let self = self else {
          return Just(
            Result.failure(ErrorType.general)).eraseToAnyPublisher()
        }

        self.error = nil
        self.loader = true

        return self.repository.getCountriesList()
      }
      .receive(on: RunLoop.main)
      .sink { [weak self] result in
        guard let self = self else { return }

        switch result {
        case .success(let item):
          self.countries = item.sorted(by: {$0.name < $1.name})
          self.baseCountries = item.sorted(by: {$0.name < $1.name})
          self.loader = false
          self.error = nil

        case .failure(let error):
          self.loader = false
          self.error = error
        }
      }
      .store(in: &cancellables)

    countriesSubject.send()
  }
```

Programski kod 4.15. Funkcija *initPipelines*

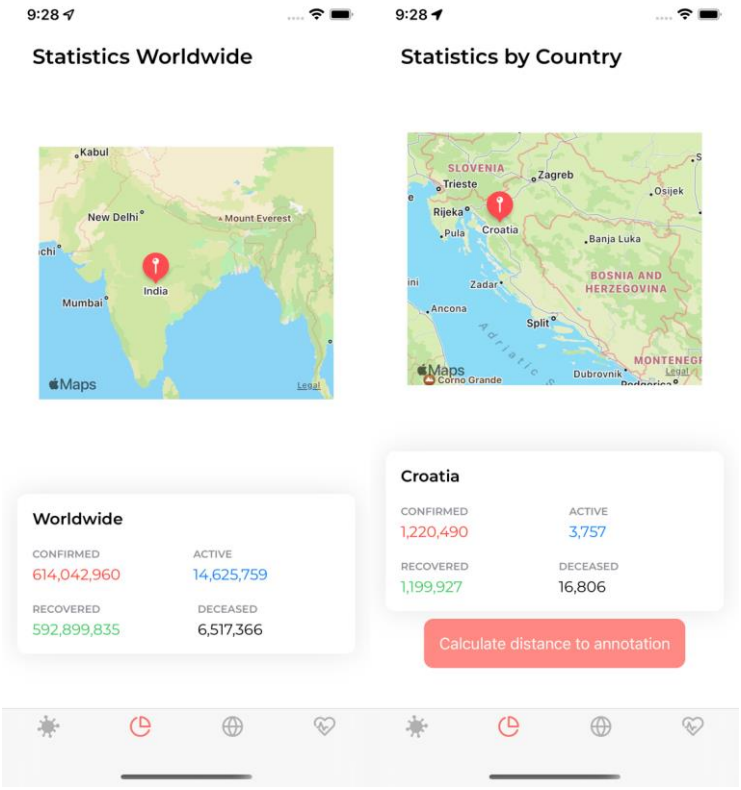
CountrySelectionView pogled sadrži *TextField* u kojem je moguće filtrirati države na temelju unesenog teksta. Pogled sadrži i listu država, te pritiskom na neku državu se mijenja globalna *useCase* varijabla na temelju odabrane države. Slika 4.5. prikazuje izgled *CountrySelectionView* pogleda.



Sl. 4.5. Izgled *CountrySelectionView* pogleda

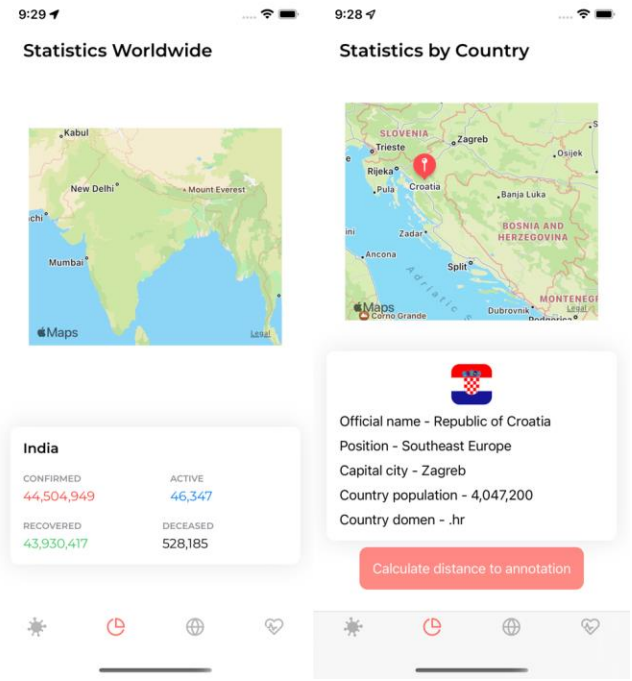
4.4. Zaslon s geografskom kartom

MapStatisticsView pogled sadrži geografsku kartu i pravokutnik u kojem se nalaze podaci vezani za slučajeve koronavirusa. Ako globalna varijabla *useCase* sadrži vrijednost *worldwide* na karti je prikazana država s najvećim brojem slučajeva koronavirusa. Unutar granica države se nalazi oznaka. Oznake se nalaze i na druge dvije države s najviše slučajeva koronavirusa. Korisniku je omogućeno kretanje po karti. Slika 4.6. prikazuje izgled *MapStatisticsView* pogleda u *worldwide* i *country* slučaju.



Sl. 4.6. Izgled *MapStatisticsView* pogleda u oba *useCase* slučaja

Pravokutnik sadrži podatke s brojevima ukupnih slučajeva koronavirusa u svijetu. Pritiskom na jednu od oznaka na karti podaci se ažuriraju, te su tada prikazani podaci o slučajevima koronavirusa vezani za državu kod koje je pritisnuta oznaka. Ponovnim pritiskom na kartu podaci su opet ažurirani i prikazuje ukupne vrijednosti slučajeva u svijetu. Ako je vrijednost globalne varijable *useCase country* na mapi je prikazana država koja je vrijednost globalne varijable. Pravokutnik sadrži podatke o koronavirus slučajevima vezanih za izabranu državu. Na mapi je prikazana oznaka, te se pritiskom na nju u pravokutniku prikazuju podaci za državu poput glavnog grada, broja stanovnika države, područje u kojemu se država nalazi itd. U *country useCase* slučaju se nalazi i gumb, te se njegovim pritiskom otvara nativna *Maps* aplikacija i računa se udaljenost od korisnikove lokacija do oznake na mapi. Prilikom otvara *Maps* aplikacije od korisnika se traži dozvola za dopuštenjem dijeljenja njegove lokacije. Slika 4.7. prikazuje *MapStatisticsView* pogled u oba slučaja kada je oznaka pritisnuta.



Sl. 4.7. *MapStatisticsView* pogled u slučaju pritisnute oznake

Programski kod 4.16 prikazuje pogled *MapStatisticsView*. Pogled se sastoji od strukture *MapView* kojom se prikazuje geografska karta koja sadrži oznake. Funkcija *statistics* služi za stvaranje pravokutnika unutar kojeg se nalaze podaci o slučajevima koronavirusa. Elementi pravokutnika su raspoređeni koristeći *VStack* i *HStack*. Funkcija prikazuje podatke dobivene iz objekta *domainItem*, tipa *MapDomainItem*, koji se nalazi unutar klase *MapViewModel*.

```
VStack {
                    Spacer()

                    MapView(viewModel: viewModel)
                       .frame(width: geo.size.width * 0.8, height: geo.size.height * 0.4)
                       .padding()

                    Spacer()
                    Spacer()
                    statistics(title: title)
                    if !(viewModel.useCase == .worldwide) {
                       Button {
                          viewModel.googleMapsButtonAction()
                       } label: {
                          Text("Calculate distance to annotation")
                             .coronaVirusAppButtonStyle()
                       }
                    }

                    Spacer()
                 }
                 .padding()
                 .addAppThemeBackground()
                 .navigationBarBackButtonHidden(true)
                 .navigationBarTitle("", displayMode: .inline)
                 .padding()
           }
```

Programski kod 4.16. Pogled *MapStatisticsView*

Podaci koji se prikazuju na *MapStatisticsView* pogledu kreirani su u klasi *MapViewModel*. Klasa koristi *StatisticsRepository* repozitorij koji je korišten i unutar *HomeViewModel* klase, te se uz objekt tipa *UseCaseSelection* predaje kao parametar konstruktoru. Klasa sadrži i subjekt izdavača podataka koji je tipa *CurrentValueSubject.* Programskim kodom 4.17 prikazan je konstruktor i svojstva klase *MapViewModel*.

```
private let repository: StatisticsRepository

    @Published var error: ErrorType?
    @Published var domainItem: MapDomainItem?
    @Published var countryDetails: CountryDetailsResponseItem?
    @Published var useCase: UseCaseSelection
    @Published var loader = true
    @Published var isShowingCountrySelection = false
    @Published var isShowingCountryDetails = false
    var annotation: MKPointAnnotation = MKPointAnnotation()

    var countryName = ""

    private let useCaseSelectionSubject: CurrentValueSubject<UseCaseSelection, Never>

    private var cancellables: Set<AnyCancellable> = .init()
    private let locationManager = LocationManager()

    init(repository: StatisticsRepository, useCase: UseCaseSelection){
        self.repository = repository
        self.useCase = useCase
        useCaseSelectionSubject = CurrentValueSubject<UseCaseSelection,
Never>.init(useCase)
        initPipelines()
    }
```

Programski kod 4.17. Konstruktor i svojstva klase *MapViewModel*

Unutar funkcije *initPipelines* se pozivaju Combine operatori na subjekt izdavaču *useCaseSelectionSubject* tipa *CurrentValueSubject*. Operator *flatMap* unutar zatvorenog izraza prima vrijednost varijable *useCase*, zatim se koristi *switch* grananje, te se poziva određena funkcija (*countryPipeline* ili *worldwidePipeline*) na temelju vrijednosti varijable. Operator vraća izdavača tipa *AnyPublisher<MapDomainItem, ErrorType>*. Funkcija *countryPipeline* kao parametar prima String koji predstavlja ime države za koju se pokušavaju dohvatiti podaci. Unutar funkcije se pozivaju repozitorij funkcije *getCountryData* kojoj se se kao parametar predaje ime države i *getVaccoidData*. Vrijednosti funkcija se spremaju u *firstPublisher* i *secondPublisher* konstante. Koristeći *Publishers.Zip* konstante se spajaju, te se koristeći *switch* grananje vrijednosti konstanti iz *successa* spremaju u nove varijable koje se kao parametri predaju *MapDomainItem* klasi. Funkcija *countryPipeline* prikazana je programskim kodom 4.18.

```
private func countryPipeline(name: String) -> AnyPublisher<Result<MapDomainItem,
ErrorType>, Never> {
    let firstPublisher = repository.getCountryData(for: name)
    let secondPublisher = repository.getVaccoidCovidData()

    return Publishers.Zip(firstPublisher, secondPublisher)
      .tryMap { data -> Result<MapDomainItem, ErrorType> in

        var countryResponseItem: [CountryDayOneResponseItem]
        var vaccoidData: VaccoidCoronaResponseItem?

        switch data.0 {
        case .success(let countryResponse):
          countryResponseItem = countryResponse
          switch data.1 {
          case .success(let vaccoid):
            for country in countryResponse {
              for item in vaccoid {
                if country.name == item.country {
                  vaccoidData = item
                  break
                }
              }
            }
          case .failure(let error):
            return Result.failure(error)
          }
        case .failure(let error):
          return Result.failure(error)
        }

        do {
          if let vaccoidData = vaccoidData {
            return try Result.success(MapDomainItem(item: countryResponseItem,
vaccoidData: vaccoidData))
          } else {
            return Result.failure(ErrorType.general)
          }
```

Programski kod 4.18. Funkcija *countryPipeline*

Unutar funkcije *worldwidePipeline* se poziva repozitorij funkcija *getWorldwideData*. Zbog toga što je u slučaju *worldwide* potrebno dohvatiti podatke za tri države s najviše koronavirus slučajeva, unutar *flatMap* operatora se poziva funkcija *getMapDomainPublisher*. Programski kod 4.19 prikazuje funkciju *worldwidePipeline*.

```
private func worldwidePipeline() -> AnyPublisher<Result<MapDomainItem,
ErrorType>, Never> {
      repository
         .getWorldwideData()
         .flatMap { [weak self] result -> AnyPublisher<Result<MapDomainItem,
ErrorType>, Never> in
            guard let self = self else {
               return Just(Result.failure(ErrorType.general)).eraseToAnyPublisher()
            }

            switch result {

            case .success(let item):
               return self.getMapDomainPublisher(item: item)

            case .failure(let error):
               return Just(Result.failure(error)).eraseToAnyPublisher()
            }
         }
         .eraseToAnyPublisher()
   }
```

Programski kod 4.19. Funkcija *worldwidePipeline*.

Unutar *getMapDomainPublisher* funkcije se kreiraju tri izdavača koji sadrže vrijednosti funkcije *getCountryData* pozvane iz repozitorija. Svakoj funkciji se predaje parametar koji predstavlja ime države kako bi funkcija mogla dohvatiti podatke vezane za tu državu. Uz tri navedena izdavača podataka, stvara se i izdavač podataka dobiven iz repozitorij funkcije *getVaccoidData*. Nakon toga se koristi operator *Publishers.Zip4* koji spaja četiri kreirana izdavača, te čeka sve dok svaki izdavač ne završi za dohvaćanjem podataka. Kreirane su i varijable *newData* i *vaccoidData* koje predstavljaju nizove u koje će biti spremljeni podaci nakon što su dohvaćeni s interneta. Nakon što su podaci dohvaćeni, zatvoreni izraz operatora *tryMap* sadrži četiri vrijednosti koje su dobivene iz *Publishers.Zip4* operatora. Vrijednosti se dodavaju u kreirane nizove, te se šalju kao parametri konstruktoru klase *MapDomainItem* koja će ekstraktirati potrebne podatke za prikaz na pogledu. U slučaju pogreške prilikom dohvaćanja podataka s interneta funkcija vraća *ErrorType* tipa *general*. Programski kod 4.20 prikazuje funkciju *getMapDomainPublisher*.

```
private func getMapDomainPublisher(item: WorldwideResponseItem) ->
AnyPublisher<Result<MapDomainItem, ErrorType>, Never> {
      if let firstCountryName = item.countries[0].countryName,
        let secondCountryName = item.countries[1].countryName,
        let thirdCountryName = item.countries[2].countryName
      {
        let firstCountry = self.repository.getCountryData(for: firstCountryName)
        let secondCountry = self.repository.getCountryData(for: secondCountryName)
        let thirdCountry = self.repository.getCountryData(for: thirdCountryName)
        let vaccoid = self.repository.getVaccoidCovidData()

        return Publishers.Zip4(firstCountry, secondCountry, thirdCountry, vaccoid)
          .tryMap { [weak self] data -> Result<MapDomainItem, ErrorType> in

            var newData: [CountryDayOneResponseItem] = []
            var vaccoidData: [VaccoidCoronaResponseItem] = []

            guard let self = self,
                let firstCountryData = self.getCountryData(data.0),
                let secondCountryData = self.getCountryData(data.1),
                let thirdCountryData = self.getCountryData(data.2)
            else {
              return Result.failure(ErrorType.general)
            }
            newData.append(firstCountryData)
            newData.append(secondCountryData)
            newData.append(thirdCountryData)

            switch data.3 {
            case .success(let item):
              vaccoidData = item
            case .failure(let error):
              return Result.failure(error.asErrorType)
            }
            do {
              return try Result.success(MapDomainItem(items: newData, vaccoidData:
vaccoidData))
            }
            catch let error {
              return Result.failure(error.asErrorType)
            }
          }
          .mapError { _ in ErrorType.general}
          .catch { Just<Result<MapDomainItem, ErrorType>>(.failure($0)) }
```

Programski kod 4.20. Funkcija *getMapDomainPublisher*

Funkcije *countryPipeline* i *worldwidePipeline* vraćaju izdavača koji se prosljeđuje idućem operatoru pozvanom na *useCase* izdavaču unutar *initPipelines* funkcije. Ako je došlo do pogreške prilikom pripremanja podataka za pogled, operator *sink* će postaviti vrijednost pogreške, koja je tipa *ErrorType*, u varijablu *error* koja će se prikazati na pogledu. U slučaju uspješnog pripremanja podataka podaci će se postaviti u varijablu *domainItem* koja se poziva u *MapStatisticsView* pogledu. Programski kod 4.21 prikazuje *useCaseSelectionSubject* izdavača podataka unutar *initPipelines* funkcije.

```
useCaseSelectionSubject
        .removeDuplicates()
        .flatMap { value -> AnyPublisher<Result<MapDomainItem, ErrorType>, Never> in
          switch value {
          case let .country(name):
            self.countryName = name
            return self.countryPipeline(name: name)
          case .worldwide:
            return self.worldwidePipeline()
          }
        }
        .receive(on: RunLoop.main)
        .sink { [weak self] response in
          guard let self = self else { return }

          switch response {
          case .success(let item):
            self.domainItem = item
            self.loader = false
            self.error = nil

          case .failure(let error):
            self.loader = false
            self.error = error
          }
        }
        .store(in: &cancellables)
```

Programski kod 4.21. Subjekt izdavač podataka *useCaseSelectionSubject*

Klasa *MapDomainItem* kao i klasa *HomeDomainItem* implementira protokol *StatisticsDomainItem*. *MapDomainItem* sadrži i svojstva *countryCoordinates* i *worldwideItems*. Svojstvo *countryCoordinates* sadrži lokacije na kojoj će biti prikazana oznaka države, dok *worldwideItems* sadrži podatke o tri države s najvećim brojem slučajeva koronavirusa. Klasa sadrži funkcije

setCountryStats i *setWorldwideStats* koje obavljaju istu funkcionalnost za postavljanje podataka koji će biti prikazani na pogledu kao i funkcije *setCountryData* i *setWorldwideData* klase *HomeDomainItem*. Klasa uz parametarski sadrži i dva konstruktora ovisno o globalnoj *useCase* varijabli koja se nalazi u *MapViewModel* klasi. Razlika između konstruktora je što u slučaju *worldwide* konstruktor prima niz objekata tipa *VaccoidCoronaResponseItem*, dok u *country* slučaju prima samo jedan objekt tipa *VaccoidCoronaResponseItem*. Programski kod 4.22 prikazuje konstruktore klase *MapDomainItem*.

```
init(title: String, confirmed: StatisticsItem, active: StatisticsItem, recovered: StatisticsItem,
death: StatisticsItem){
        super.init(
            title: title,
            confirmedCases: confirmed,
            activeCases: active,
            recoveredCases: recovered,
            deathCases: death
        )
    }

    init(item: [CountryDayOneResponseItem], vaccoidData: VaccoidCoronaResponseItem)
throws {
        super.init()
        guard let first = item.first else { throw ErrorType.general }
        title = first.name
        try setCountryStats(vaccoidData: vaccoidData)
        countryCoordinates = try getCountryCoordinates(country: first)
    }

    init(items: [CountryDayOneResponseItem], vaccoidData: [VaccoidCoronaResponseItem])
throws {
        super.init()
        title = "Worldwide"
        try setWorldwideStats(data: vaccoidData)
        worldwideItems = try getWorldwideItems(items: items, data: vaccoidData)
    }
```

Programski kod 4.22. Konstruktor klase *MapDomainItem*

Funkcija *getCountryCoordinates* dodaje u svojstvo klase *countryCoordinates* koordinate oznake koja se prikazuje na geografskoj karti vezanu za tu državu. Funkcija *getWorldwideItems* koristeći *for* petlju prolazi kroz nizove podataka dobivenih kao parametre, te stvara novu varijablu koja predstavlja niz

CountryDayOneResponseItem objekata u koji će biti spremljeni objekti koji sadrže podatke o tri države s najviše slučajeva koronavirusa. Niz se zatim koristi kao povratna vrijednost funkcije. Programskim kodom 4.23 prikazana je funkcija *getCountryCoordinates* i *getWorldwideItems*.

```
    private func getCountryCoordinates(country: CountryDayOneResponseItem) throws -
> [CountryCoordinates]{
        var newArray = [CountryCoordinates]()
        let newItem = CountryCoordinates(lat: country.lat, lon: country.lon)
        newArray.append(newItem)
        return newArray
    }

    private func getWorldwideItems(items: [CountryDayOneResponseItem]?, data:
[VaccoidCoronaResponseItem]) throws -> [CountryDayOneResponseItem] {
        guard let newItems = items else {
            throw ErrorType.general
        }

        var newWorldwideItems: [CountryDayOneResponseItem] = []
        worldwideItems = []

        for country in newItems {
            for item in data {
                if country.name == item.country {
                    newWorldwideItems.append(CountryDayOneResponseItem(
                        name: country.name,
                        lat: country.lat,
                        lon: country.lon,
                        confirmed: item.totalCases,
                        deaths: item.totalDeaths,
                        recovered: Int(item.totalRecovered) ?? 0,
                        active: item.activeCases,
                        date: country.date)
                    )
                    break
                }
            }
            if worldwideItems?.count == 3 {
                break
            } }
        return newWorldwideItems }
```

Programski kod 4.23. Funkcije *getCountryCoordinates* i *getWorldwideItems*

Geografska karta prikazana na *MapStatisticsView* pogledu stvorena je koristeći strukturu *MapView*. Struktura implementira protokol *UIViewRepresentable.* Protokol predstavlja omotača pogleda iz *UIKita* kako bi se mogao prikazati kao pogled SwiftUI-a. Struktura sadrži funkciju *makeCoordinator* koja stvara koordinatora čija je uloga rukovanje pritiskom na oznake koje se nalaze na geografskoj karti. Struktura kao parametar prima klasu *MapViewModel.* Ako globalna varijabla *useCase* sadrži vrijednost *worldwide* funkcija *updateView* poziva funkciju *setWorldwideCase*, dok u suprotnom slučaju poziva *setCountryUseCase* funkciju. Funkcije *makeCoordinator*, *updateUIView* i *makeUIView* prikazane su programskim kodom 4.24.

```
func makeUIView(context: Context) -> some MKMapView {
   return mapView
}

func makeCoordinator() -> MapManager {
   MapManager(self)
}

func updateUIView(_ uiView: UIViewType, context: Context) {
   mapView.delegate = context.coordinator
   if let domain = viewModel.domainItem {
      if domain.title == "Worldwide" {
         setWorldwideUseCase(uiView: uiView, context: context)
      }
      else {
         setCountryUseCase(uiView: uiView, context: context)
      }
   }
}
```

Programski kod 4.24. Funkcije *makeUIView*, *makeCoordinator* i *updateUIView*

Funkcija *setWorldwideCase* postavlja oznake država koje su prikazane na geografskoj karti, te koristeći funkciju *setRegion* omogućuje da geografska karta prikazuje državu s najviše slučajeva koronavirusa. Funkcija *setCountryCase* postavlja oznaku na geografskoj karti koja pomoću *setRegion* funkcije prikazuje državu koja je odabrana. Pomoću funkcije *setRegion* na geografskoj karti je postavljena država koju je potrebno prikazati. Kao parametar prima ime države koja će biti prikazana. Ime države koja se želi prikazati postavlja se svojstvu *naturalLanguageQuery*. Nakon toga se inicijalizira objekt klase *MKLocalSearch* kojoj se kao parametar predaje objekt klase *MKLocalSearch.Request*. Novi objekt zatim poziva funkciju *start* koja prikazuje državu na

geografskoj karti. Programski kod 4.25 prikazuje funkcije *setWorldwideUseCase*, *setCountryUseCase* i *setRegion*.

```
    func setWorldwideUseCase(uiView: UIViewType, context: Context) {
        if let domain = viewModel.domainItem,
           let country = domain.worldwideItems?.first?.name {
            uiView.removeAnnotations(uiView.annotations)
            setRegion(uiView: uiView, country: country)

            if let countries = domain.worldwideItems {
                for coordinate in countries {
                    let annotation = MKPointAnnotation()
                    annotation.title = coordinate.name
                    annotation.coordinate = CLLocationCoordinate2D(latitude:
Double(coordinate.lat) ?? 0.0, longitude: Double(coordinate.lon) ?? 0.0)
                    uiView.addAnnotation(annotation)
                }}}
    }

    func setCountryUseCase(uiView: UIViewType, context: Context) {
        if let domain = viewModel.domainItem,
           let country = domain.title {
            setRegion(uiView: uiView, country: country)

            let annotation: MKPointAnnotation = MKPointAnnotation()
            uiView.removeAnnotations(uiView.annotations)

            if let countries = domain.countryCoordinates {
                for item in countries {
                    annotation.coordinate = CLLocationCoordinate2D(latitude: item.lat, longitude:
item.lon)
                }
                annotation.title = domain.title
                uiView.addAnnotation(annotation)
            }
            viewModel.setAnnotation(annotation: annotation)}
    }
    private func setRegion(uiView: UIViewType, country: String) {
        searchRequest.naturalLanguageQuery = "\(country)"
        let search = MKLocalSearch(request: MKLocalSearch.Request())
        search.start { response, _ in
            guard let response = response else { return }
            uiView.setRegion(response.boundingRegion, animated: true)}
```

Programski kod 4.25. Funkcije *setWorldwideUseCase, setCountryUseCase* i *setRegion*

Klasa *MapManager* omogućuje upravljanje događajima koji slijede pritiskom na oznaku koja se nalazi na geografskoj karti. Unutar klase inicijalizira se objekt *gestureRecognizer* koji je tipa *UITapGestureRecognizer*. Klasa sadrži funkciju *mapView* koja kao parametar prima oznaku koja je pritisnuta. Ako je oznaka pritisnuta dok je slučaj *worldwide*, pravokutnik u *MapStatisticsView* pogledu prikazuje podatke o slučajevima koronavirusa za državu u kojoj je pritisnuta oznaka. Varijabli *enableTapGesture* se vrijednost postavlja u *true*, te se ponovnim pritiskom na karti poziva funkcija *tapHandler* koja poziva *initPipelines* funkciju klase *MapViewModel* kako bi se ponovno prikazivali podaci o slučajevima koronavirusa u cijelome svijetu. U slučaju *country* pritisnuta oznaka mijenja vrijednost svojstva *isShowingCountryDetails*. Programski kod 4.26 prikazuje konstruktor, svojstva i metode klase *MapManager*.

```
    let parentView: MapView
    var enableTapGesture = false
    private var gestureRecognizer = UITapGestureRecognizer()
    init(_ map: MapView) {
        self.parentView = map
        super.init()
        self.gestureRecognizer = UITapGestureRecognizer(target: self, action:
#selector(tapHandler))
        self.gestureRecognizer.delegate = self
        self.parentView.mapView.addGestureRecognizer(gestureRecognizer)
    }

    func mapView(_ mapView: MKMapView, didSelect view: MKAnnotationView) {
        if let annotation = view.annotation {
            if parentView.viewModel.domainItem?.title == "Worldwide" {
                parentView.viewModel.updateDomain(annotation: annotation)
                enableTapGesture = true
            }
            else {
                parentView.viewModel.isShowingCountryDetails.toggle()
            }}
    }

    @objc func tapHandler(_ gesture: UITapGestureRecognizer) {
        if enableTapGesture {
            parentView.viewModel.onUseCaseSelectionChange(.worldwide)
            enableTapGesture = false } }
```

Programski kod 4.26. Klasa *MapManager*

4.5. Zaslon s vijestima o koronavirusu

LatestNewsView je pogled koji prikazuje najnovije vijesti vezane za koronavirus. Pogled se sastoji od liste koja sadrži vijesti, te pritiskom na neku vijest otvara se *WebView* pogled u kojem je prikazan cijeli članak vezan za odabranu vijest. Lista prikazuje 25 vijesti, te ako se pomicanjem po zaslonu dođe do posljednje vijesti učitava se novih 25 vijesti. Pomicanjem zaslona prema dolje dolazi do ponovnog učitavanja vijesti kako bi se učitale nove vijesti ako postoje. Ponovno učitavanje novih vijesti omogućeno je strukturom *RefreshHandler*. Funkcija *list* koristi se za prikazivanje vijesti. Funkcija se nalazi unutar *ScrollView* pogleda kako bi bilo omogućeno pomicanje po zaslonu. *LatestNewsView* pogled prikazan je programskim kodom 4.27. *LatestNewsView* pogled kao parametar prima klasu *LatestNewsViewModel.*

```
ZStack(alignment: .top) {
            virusImage(blur: 3)
                .frame(width: geo.size.width * 0.2, height: geo.size.height * 0.55)
                .offset(x: geo.size.width * 0.4, y: geo.size.height * 0.1)

             virusImage(blur: 1)
                .frame(width: geo.size.width * 0.15, height: geo.size.height * 0.16)
                .offset(x: -(geo.size.width * 0.1), y: -(geo.size.height * 0.02))
                .rotationEffect(Angle(degrees: 35))

             virusImage(blur: 1)
                .frame(width: geo.size.width * 0.25, height: geo.size.height * 0.25)
                .offset(x: geo.size.width * 0.4, y: -(geo.size.height * 0.08))

              virusImage(blur: 3)
                .frame(width: geo.size.width * 0.2, height: geo.size.height * 0.25)
                .offset(x: -(geo.size.width * 0.4), y: -(geo.size.height * 0.08))

              ScrollView {
                 RefreshHandler(coordinateSpaceName: "refreshHandler") {
                    viewModel.refreshLatestNews(true)
                  }
                  list(news:news, geo: geo)
              }
              .coordinateSpace(name: "refreshHandler")
              .onAppear {
                    viewModel.handleOnAppearEvent(&UIScrollView.appearance().bounces)
              }}
```

Programski kod 4.27. Pogled *LatestNewsView*

Funkcija *list* elemente prikazuje unutar *LazyVStack* pogleda. Koristeći *ForEach* strukturu prikazane su vijesti na pogledu. Vijesti su dohvaćene iz svojstva *news* klase *LatestNewsViewModel* čiji je objekt kao parametar predan *LatestNewsView* pogledu. Programskim kodom 4.28 prikazan je *LatestNewsView* pogled. *NavigationLink* pogled je korišten kako bi se otvorio *WebView* pogled ako je vijest pritisnuta. Kao parametar prima Bool varijablu *isWebViewPresented* koja je omotana *@State* omotačem. *WebView* pogled se otvara ako je vrijednost *isWebViewPresented* varijable *true*. Jedna vijest unutar liste je prikazana koristeći *LatestNewsListItem* pogled. *ForEach* iteracijom pogledu se kao parametar predaje vrijednost iteracije koja je tipa *LatestNewsDetails*. Na pogledu je primijenjen modifikator *onTapGesture* koji omogućuje da se, pritiskom na vijest iz liste, u varijablu pogleda *url* sprema web-adresa članka vijesti koja je pritisnuta. Nakon toga se vrijednost varijable *isWebViewPresented* mijenja u *true*, te se prikazuje članak vezan za vijest. Izlaskom iz *WebView* pogleda vrijednost varijable *isWebViewPresented* se mijenja u *false*.

```
    @ViewBuilder
    private func list(news: [LatestNewsDetails], geo: GeometryProxy) -> some View {
        LazyVStack {
            ForEach(news){ item in
                NavigationLink(isActive: $viewModel.isWebViewPresented) {
                    if viewModel.isWebViewPresented {
                        WebView(url: viewModel.webViewUrl!)
                            .navigationOverride(colorScheme: colorScheme) {
                                viewModel.handleWebViewPresentation(status: .dismissed, item:
item)
                            }
                            .navigationBarTitle(Text(verbatim: .latestNewsTitle), displayMode:
.inline)
                            .addAppThemeBackground()
                    }
                } label: {
                    LatestNewsListItemView(item: item, geo: geo)
                        .padding([.leading, .trailing, .bottom])
                        .onTapGesture {
                            viewModel.handleWebViewPresentation(status: .presented, item: item)
                        }
                }
            }
```

Programski kod 4.28. Pogled *LatestNewsView*

Pogled *RefreshHandler* omogućava ponovno učitavanje vijesti prilikom pomicanja zaslona prema dolje unutar *LatestNewsView* pogleda. Pogled kao parametar prima *coordinateSpaceName*, tipa String, koja predstavlja ime koordinatnog sustava od pogleda čije će se dimenzije koristiti za prikazivanje *RefreshHandler* pogleda. Kao parametar se predaje i zatvoreni izraz *onRefresh* koji predstavlja funkcionalnost koja će se izvršiti prilikom prikaza pogleda. Programski kod 4.29 prikazuje *RefreshHandler* pogled.

```
struct RefreshHandler: View {

   var coordinateSpaceName: String
   var onRefresh: ()->Void

   @State var needRefresh: Bool = false

   var body: some View {
      GeometryReader { geo in
         if (geo.frame(in: .named(coordinateSpaceName)).midY > 50) {
            Spacer()
               .onAppear {
                  needRefresh = true
               }
         } else if (geo.frame(in: .named(coordinateSpaceName)).maxY < 10) {
            Spacer()
               .onAppear {
                  if needRefresh {
                     needRefresh = false
                     onRefresh()
                  }
               }
         }
         HStack {
            Spacer()
            if needRefresh {
               ProgressView()
            }
            Spacer()
         }
      }.padding(.top, -50)
   }
}
```

Programski kod 4.29. Pogled *RefreshHandler*

WebView pogled kao parametar prima web-adresu članka i implementira *UIViewRepresentable* protokol. Unutar funkcije *updateView* inicijalizira se objekt klase *URLRequest*, te se poziva funkcija klase *WKWebView* kako bi se otvorila web-adresa. Programski kod 4.30 prikazuje *WebView* pogled.

```
struct WebView: UIViewRepresentable {

   let url: URL
   var webView = WKWebView()

   func makeUIView(context: Context) -> WKWebView {
      return webView
   }

   func updateUIView(_ webView: WKWebView, context: Context) {
      let request = URLRequest(url: url)
      webView.load(request)
   }
}
```

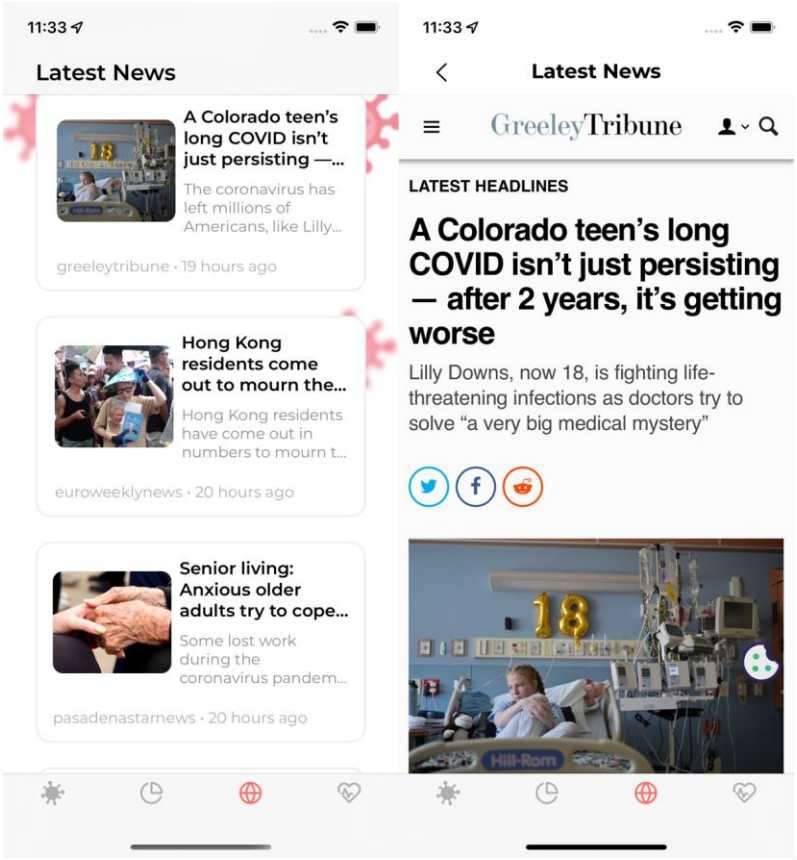
Programski kod 4.30. *WebView* pogled

Klasa *LatestNewsViewModel* koristi se za pripremanje podataka koji će biti prikazani na *LatestNewsView* pogledu. Klasa kao parametar prima objekt tipa *LatestNewsRepository* pomoću kojeg će dohvatiti potrebne podatke. Unutar funkcije *initPipelines* korištenjem *if* naredbe provjerava se vrijednost varijable *isRefreshing.* Ako je vrijednost *true,* to znači da korisnik želi ponovno učitati 25 najnovijih vijesti, pa se vrijednosti varijabli *loader* i *error* mijenjaju u *true*, odnosno *nil,* kako bi se prikazao *LoaderView* pogled i nakon njega najnovijih 25 vijesti. Ako je vrijednost *isRefreshing false*, to znači da je korisnik pomicanjem po listi došao do zadnje vijesti u listi i da je potrebno učitati novih 25 vijesti u listu. Unutar funkcije se poziva funkcija repozitorija *getLatestNews*. Ako je potrebno učitati nove vijesti u listu, vrijednost *offset* svojstva je 25, te se vrijednost povećava za 25 svaki puta kada je potrebno učitati nove vijesti. *Map* operator kao parametar prima niz vijesti tipa *LatestNewsDetails*. U svojstvo *count* se sprema veličina niza, te se svojstvo *count* koristi za provjeru je li određena vijest zadnja vijest koja je dohvaćena s interneta. Ako nije došlo do nikakve pogreške tijekom korištenja Combine operatora unutar *sink* operatora se niz koji sadrži vijesti kao parametar šalje funkciji *handleResult*. Kao parametar se šalje i vrijednost Bool varijable *isRefreshing.* Programski kod 4.31 prikazuje funkciju *initPipelines*.

```
private func initPipelines() {
      shouldRefreshSubject
         .flatMap { [weak self] value -> AnyPublisher<Result<(LatestNewsResponseItem,
Bool), ErrorType>, Never> in
            guard let self = self else {
               return Just(Result.failure(ErrorType.general)).eraseToAnyPublisher()
            }

            if value {
               self.loader = true
               self.error = nil
            }

            return self.getLatestNewsPublisher(value)
         }
         .receive(on: RunLoop.main)
         .map { [weak self] response -> Result<([LatestNewsDetails], Bool), ErrorType> in
            guard let self = self else {
               return Result.failure(ErrorType.general)
            }
            switch response {
            case .success(let item):
               self.count = item.0.pagination.total
               return Result.success((self.removeDuplicates(news: item.0.data), item.1))

            case.failure(let error):
               return Result.failure(error)
            }
         }
         .sink { [weak self] result in
            guard let self = self else { return }
            switch result {
            case .success(let item):
               self.news = self.handleResult(result: item.0, isRefreshing: item.1)
               self.loader = false
               self.error = nil
            case .failure(let error):
               self.loader = false
               self.error = error
            }
         }
         .store(in: &cancellables)
   }
```

Programski kod 4.31. Funkcija *initPipelines*

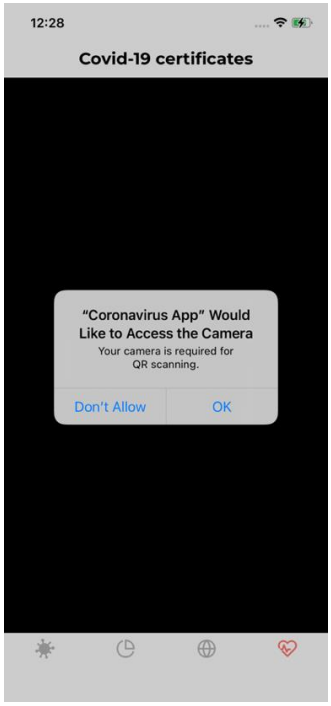
Slika 4.8 prikazuje izgled *LatestNewsView* i *WebView* pogleda. Vraćanjem nazad, odnosno zatvaranjem *WebView* pogleda, ponovno se prikazuje *LatestNewsView* pogled. Vijesti koje su prikazane na zaslonu dohvaćene su s različitih stranica poput BBC, CNN itd.



Sl. 4.8. *LatestNewsView* i *WebView* pogled

4.6. Zaslon za skeniranje COVID digitalnih potvrda

COVID digitalne potvrde sadrže QR (eng. *Quick response*) kod koji je potrebno skenirati kako bi se dobile informacije o potvrdi o cijepljenju, potvrdi o preboljenju ili negativnom rezultatu PCR testa. Zbog toga što je potrebno skenirati QR kod kako bi se dobile informacije o potvrdi, napravljena je klasa *ScanView* koja omogućuje pokretanje kamere na uređaju kako bi se mogao skenirati QR kod. Konstruktor klase kao parametar prima objekt klase *CovidCertificatesViewModel* i objekt enumeracije *ColorScheme*. Klasa sadrži privatnu varijablu *captureSession* koja je tipa *AVCaptureSession* i upravlja aktivnošću snimanja i koordinira protok podataka. Funkcija *checkCameraAuthorizationStatus* je funkcija koja omogućuje aplikaciji korištenje kamere uređaja. Funkcija prvo provjerava je li dozvola o korištenju kamere već napravljena, te ako dozvola postoji poziva se *setupCamera* funkcija. Ako dozvole nema korisniku se prikazuje obavijest koju prikazuje slika 4.9 prilikom koje može dopustiti korištenje kamere. Programski kodom 4.32 prikazana je *checkCameraAuthorizationStatus* funkcija.



Sl. 4.9. Obavijest o dozvoli kamere za skeniranje COVID digitalnih potvrda

```
private func checkCameraAuthorizationStatus(_ uiView: CameraPreview) {
    let cameraAuthorizationStatus = AVCaptureDevice.authorizationStatus(for: .video)
    if cameraAuthorizationStatus == .authorized {
        #if targetEnvironment(simulator)
            uiView.createSimulatorView(colorScheme: colorScheme)
        #else
            setupCamera(uiView)
        #endif
    } else {
        AVCaptureDevice.requestAccess(for: .video) { [weak self] granted in
            guard let self = self else {return}
            DispatchQueue.main.sync {
                if granted {
                    #if targetEnvironment(simulator)
                        uiView.createSimulatorView(colorScheme: self.colorScheme)
                    #else
                        self.setupCamera(uiView)
                    #endif
                } else {
                    uiView.createCameraDeniedView(colorScheme: self.colorScheme)
                }
            }
        }
    }
}
```

Programski kod 4.32. Funkcija *checkCameraAuthorizationStatus*

Unutar funkcije *setupCamera* se stvara objekt klase *AVCaptureDevice* kojeg je potrebno odmotati. Kao ulazna vrijednost postavlja se *video*, te se stvara varijabla *captureDeviceInput* koja predstavlja ulaz koji se koristi za snimanje korištenjem kamere. Varijabla *captureMetadataOutput* predstavlja izlazne vrijednosti dobivene kamerom, a te izlazne vrijednosti mogu biti dobivene iz *qr* tipa objekata. Unutar funkcije se poziva funkcija *setupPreviewLayer* koja stvara varijablu *previewLayer*, tipa *AVCaptureVideoPreviewLayer*, koja omogućuje prikazivanje videa na zaslonu uređaja. Varijabla *previewLayer* postavljena je kao *previewLayer* objekta *uiView* koji je predan kao parametar funkciji *setupPreviewLayer*. Funkcija *updateUIView* omogućuje zaustavljanje rada kamere kada je COVID digitalna potvrda skenirana. Programski kod 4.33 prikazuje funkcije *setupCamera*, *setupPreviewLayer* i *updateUIView*.

```
    func setupCamera(_ uiView: CameraPreview){
        if let captureDevice = AVCaptureDevice.default(for: AVMediaType.video),
           let captureDeviceInput = try? AVCaptureDeviceInput(device: captureDevice) {

            if captureSession.canAddInput(captureDeviceInput) {
                captureSession.addInput(captureDeviceInput)
            }

            let captureMetadataOutput = AVCaptureMetadataOutput()

            if captureSession.canAddOutput(captureMetadataOutput){
                captureSession.addOutput(captureMetadataOutput)
                captureMetadataOutput.metadataObjectTypes = [.qr]
                captureMetadataOutput.setMetadataObjectsDelegate(self, queue:
DispatchQueue.main)
            }

            setupPreviewLayer(uiView: uiView)
        }
    }

    private func setupPreviewLayer(uiView: CameraPreview){
        let previewLayer = AVCaptureVideoPreviewLayer(session: captureSession)
        previewLayer.videoGravity = AVLayerVideoGravity.resizeAspectFill

        uiView.layer.addSublayer(previewLayer)
        uiView.layer.shadowPath = UIBezierPath(rect: uiView.bounds).cgPath
        uiView.previewLayer = previewLayer
    }

    func updateUIView(_ uiView: CameraPreview, context:
UIViewRepresentableContext<ScanView>) {
        if handler.isRescan {
            captureSession.stopRunning()
        } else {
            setupCamera(uiView)
            captureSession.startRunning()
        }
     }
    }
```

Programski kod 4.33. Funkcije *setupCamera*, *setupPreviewLayer* i *updateUIView*

Klasa *ScanView* implementira protokol *AVCaptureMetadataOutputObjectsDelegate* i njegovu funkciju *metadataOutput* koji se poziva nakon uspješnog skeniranja QR koda. Nakon što je QR kod skeniran rad kamere je zaustavljen. Dohvaćena vrijednost je tipa *AVMetadataObject*. Nakon toga se stvara varijabla *readableObject* koja je tipa *AVMetadataMachineReadableCodeObject* kako bi se mogla dobiti String vrijednost QR koda koja se sprema u varijablu *stringValue*, te se poziva funkcija *handleScannedValue* klase *CovidCertificatesViewModel* kako bi se dekodirala String vrijednost. Programski kod 4.34 prikazuje funkciju *metadataOutput*.

```
    func metadataOutput(_ output: AVCaptureMetadataOutput, didOutput metadataObjects:
[AVMetadataObject], from connection: AVCaptureConnection) {
        captureSession.stopRunning()

        if let metadataObject = metadataObjects.first {
            guard let readableObject = metadataObject as?
AVMetadataMachineReadableCodeObject else { return }
            guard let stringValue = readableObject.stringValue else { return }
            handler.handleScannedValue(stringValue)
        }
    }
```

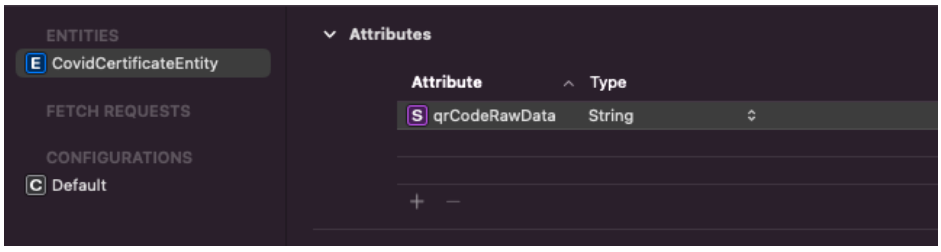
Programski kod 4.34. Funkcija *metadataOutput*

Klasa *CovidCertificateDecoder* je klasa koja omogućuje dekodiranje String vrijednosti dobivene iz QR koda kako bi se dobile informacije iz COVID digitalne potvrde. Dekodiranje se obavlja unutar funkcije *decodeData* koja kao parametar prima String vrijednost. Dekodiranje je omogućeno korištenjem biblioteke *EUDCC*. Korištenjem *Just* kreira se novi izdavač podataka iz ulaznog parametra funkcije. Vrijednost izdavača se, korištenjem *map* operatora, predaje funkciji *decode* klase *EUDCCDecoder* koja je implementirana iz *EUDCC* i *EUDCCDecoder* biblioteka. Dekodirana vrijednost se sprema u varijablu *decodingResult*, te ako je dekodiranje uspješno kreira se *Result* enumeracija koja sadrži uspješne i neuspješne vrijednosti i dekodirana vrijednost se sprema u uspješnu vrijednost. Ako je dekodiranje neuspješno stvara se *Result* enumeracija s *ErrorType.empty* vrijednosti kao vrijednost neuspješne vrijednosti enumeracije. Na kraju se poziva funkcija *eraseToAnyPublisher* koja omogućuje stvaranje novog izdavača podataka. Funkcija *decodeData* vraća izdavača podataka. Programski kod 4.35 prikazuje funkciju *decodeData*.

```
class CovidCertificateDecoder {
   func decodeData(_ code: String) -> AnyPublisher<Result<EUDCC, ErrorType>,
Never> {
      Just(code)
         .map { data -> Result<EUDCC, ErrorType> in
            let decodingResult = EUDCCDecoder().decode(from: data)

            switch decodingResult {
            case .success(let eudcc):
               return Result.success(eudcc)
            case.failure(_):
               return Result.failure(ErrorType.empty)
            }
         }
         .eraseToAnyPublisher()
   }
}
```

Programski kod 4.35. Funkcija *decodeData*

Lokalno spremanje podataka omogućeno je koristeći *Core Data* programski okvir. Prije upotrebe programskog okvira potrebno je definirati entitet. Entitet je klasa predstavljena u objektnom grafu. Sadrži ime, svojstva i veze između svojstava i ostalih entiteta. Entitet je instanca *NSEntityDescription* klase. *CovidCertificateEntity* predstavlja entitet sa svojstvom *qrCodeRawData* koje omogućuje lokalno spremanje vrijednosti dobivene skeniranjem QR koda. Slika 4.10 prikazuje *CovidCertificateEntity* entitet.



Sl. 4.10. Entitet *CovidCertificateEntity*

Klasa *CoreDataManager* je klasa koja sadrži metode koje omogućuje lokalno spremanje podataka. Klasa ima samo jednu instancu, odnosno klasa predstavlja *singleton*. U konstruktoru klase, koji je

privatan, u svojstvo klase *container* se postavlja vrijednost spremnika u koji će se podaci lokalno spremati. Zatim se poziva funkcija *loadPersistentStores* koja omogućuje da spremnik trajno pohrani podatke i dovrši stvaranje skupa podataka. Programskim kodom 4.36 prikazana su svojstva i konstruktor klase *CoreDataManager*.

```
static let shared: CoreDataManager = CoreDataManager()

    private let container: NSPersistentContainer

    private init(){
        container = NSPersistentContainer(name: "CovidCertificatesContainer")
        container.loadPersistentStores { description, error in
            if let error = error {
                print(error.localizedDescription)
            }
        }
    }
```

Programski kod 4.36. Svojstva i konstruktor klase *CoreDataManager*

Metoda *getCertificates* je metoda koja omogućuje dohvaćanje lokalno spremljenih COVID digitalnih potvrda. Konstanta *request* je *NSFetchRequest* tipa podatka i ona predstavlja uvjet kojeg će se tipa entiteta dohvatiti podaci iz lokalne memorije. Nakon toga se poziva funkcija *fetch* koja omogućuje dohvaćanje potrebnih vrijednosti. Metoda *saveData* omogućuje spremanje promjena koje su nastale u lokalnoj memoriji. Metodom *addCertificate* se COVID digitalne potvrde dodaju u lokalnu memoriju. Metoda kao parametar prima String vrijednost koja predstavlja vrijednost skeniranu QR kodom. Unutar metode se poziva metoda *checkDuplicateCertificate* koja provjerava postoji li već ta vrijednost u lokalnoj memoriji. Ako ne postoji, stvara se novi objekt tipa *CovidCertificateEntity*, te se poziva metoda *saveData*. Metoda *deleteCertificate* je metoda kojom se briše COVID digitalna potvrda iz lokalne memorije. Metoda kao parametar prima String vrijednost QR koda koji se želi izbrisati iz lokalne memorije. Unutar metode se poziva metoda *getCertificates* kako bi se dohvatile spremljene potvrde, te se koristeći *for* petlju prolazi kroz potvrde kako bi se pronašla potvrda koja se želi izbrisati. Programski kod 4.37 prikazuje metode *CoreDataManager* klase.

```
    func getCertificates() -> [CovidCertificateEntity] {
        let request = NSFetchRequest<CovidCertificateEntity>(entityName:
"CovidCertificateEntity")
        do {
            return try container.viewContext.fetch(request)
        } catch let error {
            print(error)
            return []
        }
    }
    func addCertificate(_ value: String){
        if checkDuplicateCertificate(value){
            let newCertificate = CovidCertificateEntity(context: container.viewContext)
            newCertificate.qrCodeRawData = value
            saveData()
        }
    }
    func deleteCertificate(_ value: String){
        let certificates = getCertificates()
        for certificate in certificates where certificate.qrCodeRawData == value {
            container.viewContext.delete(certificate)
        }
        saveData()
    }

    private func checkDuplicateCertificate(_ value: String) -> Bool {
        let request = NSFetchRequest<CovidCertificateEntity>(entityName:
"CovidCertificateEntity")
        do {
            let certificates = try container.viewContext.fetch(request)
            if certificates.contains(where: { $0.qrCodeRawData == value }){
                return false
            }
        }
        return true
    }
    private func saveData(){
        do {
            try container.viewContext.save()
        } catch let error {
            print(error)
        }
    }
```

Programski kod 4.37. Metode *CoreDataManager* klase

Klasa *CovidCertificatesViewModel* je klasa koja sadrži podatke koji će biti prikazani na zaslonu mobilnog uređaja. Konstruktor klase kao parametar prima *CovidScannerRepository* objekt s kojim je omogućeno dekodiranje COVID digitalnih potvrda. Izdavači podataka *certificatesListPublisher* i *scannedCertificatePublisher* predstavljaju subjekt izdavače podataka. *CertificatesListPublisher* je tipa *CurrentValueSubject* i on se koristi za prikaz lokalno spremljenih COVID digitalnih potvrda. *ScannedCertificatePublisher*, koji je tipa *PassthroughSubject*, prosljeđuje skeniranu COVID digitalnu potvrdu repozitorij funkciji *getCertificateDetails* kako bi se potvrda mogla dekodirati. Konstruktor i svojstva klase *CovidCertificatesViewModel* prikazani su programskim kodom 4.38.

```
    let repository: CovidScannerRepository

    @Published var allCertificates: [CovidCertificateDomainItem]?
    @Published var validCertificates: [CovidCertificateDomainItem]?
    @Published var data: CovidCertificateDomainItem?

    @Published var isScanViewPresented = false
    @Published var isDetailSheetPresented = false
    @Published var isErrorSheetPresented = false
    @Published var isAlertPresented = false
    @Published var selectedCertificates: CertificatesListType = .allCertificates
    private var qrCodeRawData = ""

    private let certificatesListPublisher = CurrentValueSubject<[CovidCertificateEntity],
Never>
        .init(CoreDataManager.shared.getCertificates())

    private var scannedCertificatePublisher = PassthroughSubject<String, Never>()

    private var cancellables: Set<AnyCancellable> = .init()

    init(repository: CovidScannerRepository){
        self.repository = repository
        initPipelines()
    }
```

Programski kod 4.38. Konstruktor i svojstva klase *CertificatesViewModel*

Funkcija *initPipelines* je funkcija koja se poziva prilikom kreiranja objekta *CovidCertificatesViewModel* klase. Izdavač podataka *scannedCertificatePublisher* skenirane COVID digitalne potvrde prosljeđuje funkciji *handleValue* u kojoj se potvrda sprema u lokalnu memoriju, te

se šalje zahtjev *certifiatesListPublisher* izdavaču podataka kako bi se ažurirala varijabla *allCertificates.* Programski kod 4.39 prikazuje *scannedCertificatePublisher* izdavača podataka.

```
scannedCertificatePublisher
        .flatMap { [weak self] value -> AnyPublisher<Result<EUDCC, ErrorType>, Never> in
            self?.handleValue(value)
            guard let publisher = self?.repository.getCertificateDetails(value) else {
                return Empty().eraseToAnyPublisher()
            }
            return publisher
        }
        .map { $0 }
        .sink(receiveCompletion: { _ in },
              receiveValue: { [weak self] result in
            switch result {
            case .success(let result):
                do {
                    if let qrCode = self?.qrCodeRawData {
                        self?.data = try CovidCertificateDomainItem(eudcc: result, qrCodeRawData:
qrCode)
                        self?.isDetailSheetPresented = true
                    }
                }
                catch {
                    self?.isErrorSheetPresented = true
                }
            case .failure(_):
                self?.isErrorSheetPresented = true
            }
        })
        .store(in: &cancellables)
```

Programski kod 4.39. Izdavač podataka *scannedCertificatePublisher*

Unutar *initPipelines* funkcije izdavač podataka *certificatesListPublisher* dohvaća lokalno spremljene COVID digitalne potvrde i sprema ih u varijablu *allCertificates*. Digitalne potvrde se prosljeđuju i funkciji *setValidCertificates* u kojoj se digitalne potvrde filtriraju s uvjetom da potvrde budu ispravne. Ako je dohvaćanje podataka neuspješno, vrijednost *allCertificates* se postavlja kao *nil*. Programski kod 4.40 prikazuje izdavača podataka *certificatesListPublisher*.

```
        certificatesListPublisher
            .map { [weak self] result -> Result<[CovidCertificateDomainItem], ErrorType>
in
                guard let certificates = self?.getLocalCertificates(result) else {
                    return Result.failure(ErrorType.empty)
                }
                return Result.success(certificates)
            }
            .sink { _ in } receiveValue: { [weak self] result in
                switch result {
                case .success(let certificates):
                    self?.allCertificates = certificates
                    self?.setValidCertificates(certificates)
                case .failure(_):
                    self?.allCertificates = nil
                }
            }
            .store(in: &cancellables)
```

Programski kod 4.40. Funkcija *initPipelines*

Funkcija *getLocalCertificates* je funkcija koja se poziva unutar izdavača podataka *certificatesListPublisher*. Kao parametar prima niza podataka tipa *CovidCertificateEntity* i kao povratnu vrijednost vraća niza podataka tipa *CovidCertificateDomainItem*. Unutar funkcije se koristi *for* petlja kako bi se dohvatio svaki objekt iz niza podataka dobivenih iz parametra. Dohvaćeni objekt se zatim dekodira i prosljeđuje konstruktoru klase *CovidCertificateDomainItem* kako bi se stvorio objekt koji će prikazivati podatke na zaslonu mobilnog uređaja. Dobiveni objekt se zatim dodaje u niz *newCertificates* koji će biti povratna vrijednost funkcije. Funkcija *setValidCertificates* je funkcija koja kao parametar prima niz podataka tipa *CovidCertificateDomainItem*, te iz dobivenog niza, koristeći *for* petlju, provjerava ispravnog COVID digitalne potvrde. Ako je potvrda ispravna dodaje se u varijablu *newCertificates*. Na kraju se vrijednost varijable *newCertificates* postavlja kao vrijednost svojstva *validCertificates*. Funkcija *handleValue* je funkcija koja se poziva unutar *scannedCertificatePublisher* izdavača podataka kada je COVID digitalna potvrda skenirana. Funkcija kao parametar prima String podatak i taj podatak se predaje funkciji *addCertificate* klase *CoreDataManager* kako bi se digitalna potvrda dodala u lokalnu memoriju mobilnog uređaja. Na kraju se izdavaču podataka *certificatesListPublisher* šalje vrijednost dobivena iz funkcije *getCertificates* klase *CoreDataManager* kako bi se ažurirali novi podaci iz lokalne memorije. Programski kod 4.41 prikazuje funkcije *getLocalCertificate*, *setValidCertificates i handleValue*.

```
private func getLocalCertificates(_ data: [CovidCertificateEntity]) ->
[CovidCertificateDomainItem]? {
    var newCertificates: [CovidCertificateDomainItem] = []
    for item in data {
      if let qrCodeRawValue = item.qrCodeRawData {
        repository
          .getCertificateDetails(qrCodeRawValue)
          .map { result -> CovidCertificateDomainItem? in
            switch result {
            case .success(let eudcc):
              do {
                return try CovidCertificateDomainItem(eudcc: eudcc, qrCodeRawData:
qrCodeRawValue)
              }
              catch {return nil}
            case.failure(_):
              return nil
            }
          }
          .compactMap { $0 }
          .sink { result in
            newCertificates.append(result)
          }
          .store(in: &cancellables)
      }
    }
    return newCertificates.reversed()
  }

  private func setValidCertificates(_ data: [CovidCertificateDomainItem]){
    var newCertificates: [CovidCertificateDomainItem] = []

    for value in data {
      if let isCertificationValid = value.isCertificationValid, !isCertificationValid {
        newCertificates.append(value)
      }
    }
    self.validCertificates = newCertificates
  }

  private func handleValue(_ value: String){
    qrCodeRawData = value
    CoreDataManager.shared.addCertificate(value)
    certificatesListPublisher.send(CoreDataManager.shared.getCertificates())
  }
```

Programski kod 4.41. Funkcije *getLocalCertificate, setValidCertificates* i *handleValue*

CovidCertificateDomainItem je struktura čiji objekt se nalazi unutar klase *CovidCertificatesViewModel* i pomoću njega klasa prikazuje podatke na zaslonu mobilnog uređaja. Struktura sadrži svojstva različitih tipova koji će biti korišteni za prikaz. Struktura kao parametre konstruktora prima objekt klase *EUDCC* koji predstavlja dekodirani objekt QR koda i String podatak koji predstavlja String vrijednost QR koda koji se dekodira kako bi se kreirao objekt. Unutar konstruktora se poziva funkcija *checkCertificateType* koja, iz *EUDCC* objekta dobivenog kao parametar, provjerava o kojem tipu COVID digitalne potvrde se radi. Nakon toga se koristeći *if* grananje poziva određena funkcija. Programski kod 4.42 prikazuje konstruktor i svojstva strukture *CovidCertificateDomainItem* i funkciju *checkCertificateType*.

```
    let qrCodeRawData: String
    var icon: Image?
    var title: String?
    var subtitle: String?
    var certificateTypeDetails: String?
    var certificateGeneralInfo: String?
    var activeTitle: String?
    var dateOfBirth: String?
    var country: String?
    var certificateValidationMessage: String?
    var isCertificationValid: Bool?

    init(eudcc: EUDCC, qrCodeRawData: String) throws {
        self.qrCodeRawData = qrCodeRawData
        try checkCertificateType(eudcc)
    }
    private mutating func checkCertificateType(_ eudcc: EUDCC) throws {
        if eudcc.recovery != nil {
            try setRecoveryData(eudcc, isValid:
getValidationResult(EUDCCValidator().validate(eudcc: eudcc, rule: .isRecoveryValid)))
        } else if eudcc.vaccination != nil {
            try setVaccinationData(eudcc, isValid:
getValidationResult(EUDCCValidator().validate(eudcc: eudcc, rule:
.isVaccinationExpired())))
        } else {
            try setRecoveryData(eudcc, isValid:
getValidationResult(EUDCCValidator().validate(eudcc: eudcc, rule: .isTestValid())))
        }
```

Programski kod 4.42. Svojstva i konstruktor strukture *CovidCertificateDomainItem* i funkcija *checkCertificateType*

Funkcije *setRecoveryData*, *setVaccinationData* i *setTestData* su funkcije kojima se postavljaju vrijednosti svojstvima strukture. Sve tri funkcije imaju istu funkcionalnost – postavljanje vrijednosti svojstvima strukture. Razlikuju ih podaci dobiveni iz *EUDCC* objekta. Programski kod 4.43 prikazuje funkciju *setRecoveryData*.

```
    private mutating func setRecoveryData(_ eudcc: EUDCC, isValid: Bool) throws {
        guard let certificateValidUntil = eudcc.recovery?.certificateValidUntil,
            let certificateIssuer = eudcc.recovery?.certificateIssuer,
            let firstPositiveTestResult = eudcc.recovery?.dateOfFirstPositiveTestResult,
            let testCountry = eudcc.recovery?.countryOfTest.localizedString()
        else {
            throw ErrorType.general
        }

        self.isCertificationValid = isValid
        self.certificateValidationMessage = isValid == true ? "Certificate has expired." : ""
        self.icon = Image("recovery")
        self.title = eudcc.name.formatted()
        self.subtitle = "Recovery Certificate"
        self.certificateTypeDetails = "Valid until -
\(DateFormatter.dayMonthYear.parseToString(date: certificateValidUntil))"
        self.certificateGeneralInfo = "Certificate issuer - \(certificateIssuer)"
        self.activeTitle = "First positive test - \(DateFormatter.dayMonthYear.parseToString(date:
firstPositiveTestResult))"
        self.dateOfBirth = "Date of birth - \(DateFormatter.dayMonthYear.parseToString(date:
eudcc.dateOfBirth))"
        self.country = "Test country - \(testCountry)"
    }
```

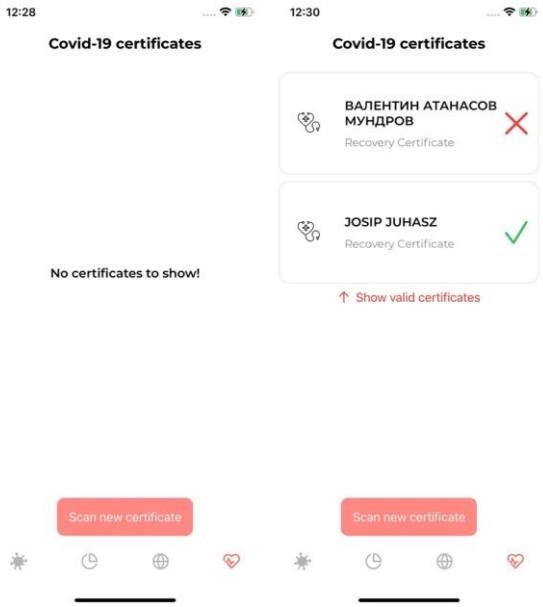
Programski kod 4.43. Funkcija *setRecoveryData*

Pogled *CovidCertificatesView* je pogled koji je prikazan kada se otvori zaslon za skeniranje COVID digitalnih potvrda. Pogled se sastoji od liste u kojoj se prikazane lokalno spremljene COVID digitalne potvrde, gumba koji omogućuje prikaz svih ili samo ispravnih potvrda i gumba pritiskom na kojeg se otvara kamera kako bi se skenirao QR kod. Pogled je stvoren koristeći *ScrollView* kako bi korisniku bilo omogućeno kretanje po zaslonu u slučaju većeg broja lokalno spremljenih COVID digitalnih potvrda. Programski kod 4.44 prikazuje pogled *CovidCertificatesView*.

```
if let certificates = viewModel.certificates {
                if !certificates.isEmpty {
                    ScrollView(showsIndicators: false) {
                        ForEach(certificates){ item in
                            CovidCertificatesListItem(data: item)
                                .onTapGesture {
                                    viewModel.handleListItemTapAction(item)
                                }
                            if item == certificates.last {
                                Button {
                                    viewModel.handleCertificatesTypeButtonAction()
                                } label: {
                                    HStack {
                                        Image(systemName: viewModel.imageName)

                                        Text(viewModel.buttonTitle)
                                    }
                                }
                            }
                        }
                    }
                    .padding()

                } else {
                    Spacer()

                    Text("No certificates to show!")
                        .commonFont(.semiBold, style: .body)
                        .padding(.bottom)

                    if let validCertificates = viewModel.validCertificates,
                      let allCertificates = viewModel.allCertificates,
                      validCertificates.isEmpty && !allCertificates.isEmpty {
                      Button {
                          viewModel.handleCertificatesTypeButtonAction()
                      } label: {
                          Text("Show all certificates")
                      }
                    }
                }
```

Programski kod 4.44. Pogled *CovidCertificatesView*

Slika 4.11 prikazuje *CovidCertificatesView* pogled na zaslonu mobilnog uređaja kada u lokalnoj memoriji nema spremljenih digitalnih potvrda i kada se potvrde nalaze u lokalnoj memoriji (ispravna i neispravna COVID digitalna potvrda).



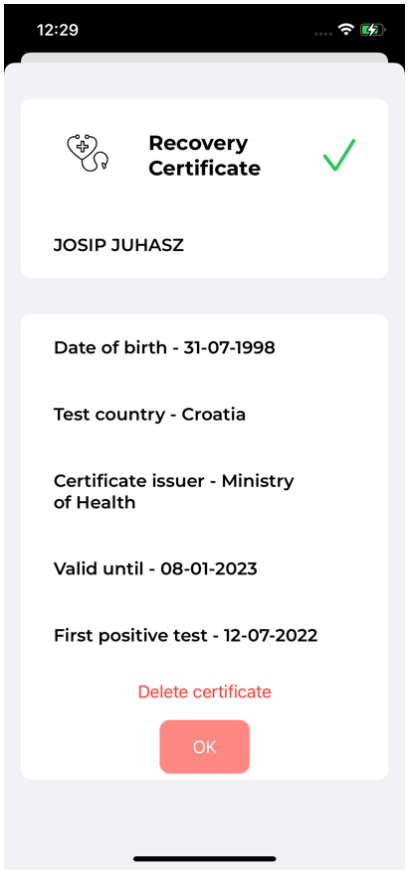
Sl. 4.11. Izgled *CovidCertificatesView* pogleda na mobilnom uređaju kada u lokalnoj memoriji ne postoje i postoje COVID digitalne potvrde

Pogled *CertificatesDetailsView* koristi se za prikaz detalja COVID digitalne potvrde. Pogled se prikazuje kada je skeniran QR kod COVID digitalne potvrde ili kada korisnik pritisne na neku potvrdu koja se nalazi na listi prikazanoj unutar *CovidCertificatesView* pogleda. Na pogledu su prikazani podaci dobiveni iz klase *CovidCertificatesViewModel*. Podaci su podijeljeni u dva odjeljka. Pogled sadrži gumb za brisanje potvrde iz lokalne memorije i gumb s kojim se ponovno prikazuje lista lokalno spremljenih potvrda. Programski kod 4.45 prikazuje pogled *CertificatesDetailsView*.

```
            Section {
                HStack(spacing: 10) {
                    icon
                        .renderingMode(.template)
                        .resizable()
                        .frame(width: UIScreen.main.bounds.width * 0.17, height:
UIScreen.main.bounds.width * 0.17)

                    Text(subtitle)
                        .commonFont(.bold, style: .title3)
                        .fixedSize(horizontal: false, vertical: true)
                        .padding(.leading)

                    Spacer()

                    certificateValidationImage(isCertificationValid)
                }
                .padding()

                listRow(text: data.title)
            }

            Section {

                listRow(text: data.dateOfBirth)

                listRow(text: data.country)

                listRow(text: data.certificateGeneralInfo)

                listRow(text: data.certificateTypeDetails)

                listRow(text: data.activeTitle)

                if isCertificationValid {
                    HStack(spacing: 10) {
                        Image(systemName: "info.circle.fill")
                            .resizable()
                            .frame(width: UIScreen.main.bounds.width * 0.08, height:
UIScreen.main.bounds.width * 0.08)

                        listRow(text: data.certificateValidationMessage)
                    }
                    .padding(.leading)
                }
```

Programski kod 4.45. *CertificatesDetailsView* pogled

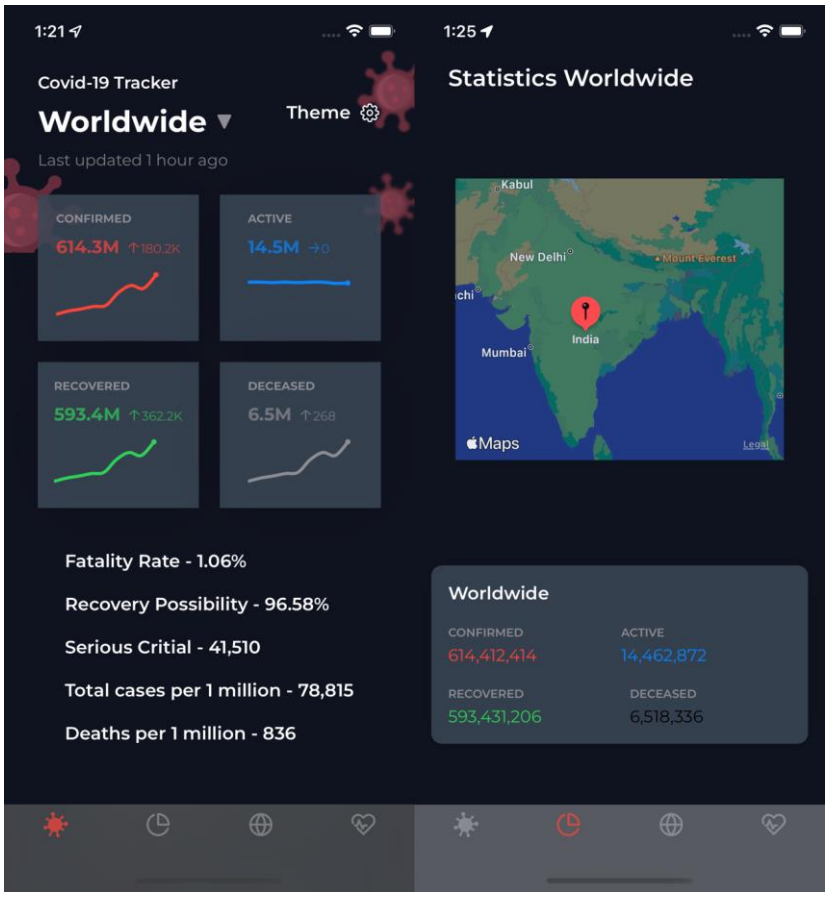
Slika 4.12 prikazuje izgled *CertificatesDetailsView* pogleda na mobilnom uređaju. Na pogledu se nalaze podaci poput tipa digitalne potvrde, ime vlasnika potvrde, datum rođenja vlasnika, datum isteka valjanosti potvrde itd. Kako je pogled *sheet*, pogled je moguće spustiti što dovodi do vraćanja na *CovidCertificatesView* pogled.



Sl. 4.12. Izgled *CertificatesDetailsView* pogleda na mobilnom uređaju

4.7. Tamni način rada aplikacije

Odabir svijetlog ili tamnog načina rada aplikacije omogućen je u *HomeView* pogledu. Pritiskom na gumb *Change Theme* prikazuju se dvije mogućnosti odabira načina rada aplikacije. Slika 4.13 prikazuje *HomeView* i *MapStatisticsView* pogled u tamnom načinu rada aplikacije.



Sl. 4.13 *HomeView* i *MapStatisticsView* pogled u tamnom načinu rada aplikacije

HomeViewModel sadrži funkciju *changeTheme* s kojom je omogućeno mijenjanje načina rada aplikacije. Funkcija kao parametar prima vrijednost enumeracije *UIUserInterfaceStyle* koja ima slučajeve *light* i *dark*. Unutar funkcije se pristupa klasi *UIWindowScene* koja upravlja instancama klasa koje omogućuju prikaz pogleda na svim zaslonima mobilne aplikacije. Funkcija *changeTheme* se poziva unutar *HomeView* pogleda kao operacija koju izvršava *Button* koji se nalazi unutar funkcije *header*. Programski kod 4.46 prikazuje funkciju *changeTheme*, a programski kod 4.47 prikazuje poziv funkcije unutar *HomeView* pogleda.

```
func changeTheme(_ theme: UIUserInterfaceStyle){
      (UIApplication.shared.connectedScenes.first as?
       UIWindowScene)?.windows.first!.overrideUserInterfaceStyle = theme
   }
```

Programski kod 4.46. Funkcija *changeTheme*

```
Menu {
            Button {
               viewModel.changeTheme(.light)
            } label: {
               Text("Light")
            }
            Button {
               viewModel.changeTheme(.dark)
            } label: {
               Text("Dark")
            }
         } label: {
            Text("Theme")
               .commonFont(.semiBold, style: .body)
               .foregroundColor(.white)

            Image(systemName: "gearshape")
               .foregroundColor(.white)
         }
```

Programski kod 4.47. Poziv funkcije *changeTheme*

5. ZAKLJUČAK

U lipnju 2022. godine u svijetu je potvrđeno više od 530 milijuna slučaja koronavirusa. Kako se virus brzo proširio po cijelome svijetu, razne epidemiološke mjere poput zatvaranja državnih granica, zabrana putovanja i zabrana javnih događaja su bile provedene. Kako bi se korisnicima omogućilo lakše praćenje svih informacija vezanih za koronavirus, ovaj rad prikazuje izradu i implementaciju iOS mobilne aplikacije u programskom jeziku Swift koja kao cilj ima omogućiti korisniku praćenje slučaja koronavirusa u svijetu, pa tako i u pojedinačnim državama dohvaćanjem informacija o slučajima s mrežne baze podataka. Osim o trenutnim brojevima slučaja, korisnik može odabrati i određeni datum za određenu državu kako bi provjerio slučajeve koronavirusa. Korisniku je omogućeno i skeniranje COVID digitalnih potvrda koje osobe dobiju nakon što su preboljele virus, kako bi imao uvid o ispravnosti potvrde. Dohvaćanjem najnovijih vijesti s mrežne baze podataka, korisnik može biti pravovremeno obaviješten o širenju koronavirusa i novim mjerama zaštite. Zahtjevi na aplikaciju rješavani su tako da je prvo osmišljen cjelokupni sustav koji se sastoji od mobilne aplikacije i mrežne baze podataka.

LITERATURA

Commented [JB12]: Dodajte najmanje 2-3 knjige o swiftu i iOS-u u literaturu i pozovite se na nju

[1] Republika Hrvatska Ministarstvo Zdravstva, CovidGO [online], Ministarstvo Zdravstva, 2021, dostupno na: https://apps.apple.com/us/app/covidgo/id1572794000 [13.9.2022]

[2] Robert Koch-Institut, Corona-Warn-App [online], Robert Koch-Institut, 2021, dostupno na: https://apps.apple.com/be/app/corona-warn-app/id1512595757[13.9.2022]

[3] Republika Hrvatska Ministarstvo Zdravstva, Stop COVID-19 [online], Ministarstvo Zdravstva, 2020, dostupno na: https://apps.apple.com/hr/app/stop-covid-19/id1519179939[13.9.2022]

[4] Department of Health and Social Care, NHS COVID-19 [online], NHS Test and Trace, Department of Health and Social Care, 2022, dostupno na: https://apps.apple.com/gb/app/nhs-covid-19/id1520427663 [13.9.2022]

[5] Health Service Executive, COVID Tracker Ireland [online], Health Service Executive, 2020, dostupno na: https://apps.apple.com/ie/app/covid-tracker-ireland/id1505596721 [13.9.2022]

[6] M. Ward, Swift Programming: The Big Nerd Ranch Guide, Big Nerd Ranch Guides, Atlanta, 2020

[7] Apple Inc., The Swift Programming Language, Functions [online], dostupno na https://docs.swift.org/swift-book/LanguageGuide/Functions.html [1.8.2022]

[8] Apple Inc., The Swift Programming Language, Closures [online], dostupno na https://docs.swift.org/swift-book/LanguageGuide/Closures.html [1.8.2022]

[9] Apple Inc., The Swift Programming Language, Closures [online], dostupno na https://docs.swift.org/swift-book/LanguageGuide/ClassesAndStructures.html [1.8.2022]

[10] Apple Inc., The Swift Programming Language, Closures [online], dostupno na https://docs.swift.org/swift-book/LanguageGuide/Protocols.html [1.8.2022]

[11] AppCoda, Learn Swift-UI , Swift-UI Basics [online], dostupno na https://www.appcoda.com/learnswiftui/swiftui-basics.html [1.8.2022]

[12] S.Mishali, F. Pillet and M.Todorov, Combine: Asynchronous Programming with Swift, Razeware LLC, USA, 2021

[13] J. Heck, Using Combine, Gumroad, USA, 2020

[14] Dino Bartošak, Swift Tutorial Introduction To MVVM [online], Toptal, dostupno na https://www.toptal.com/ios/swift-tutorial-introduction-to-mvvm [1.8.2022]
[15] Apple Inc., X-Code [online], dostupno na https://developer.apple.com/xcode/ [1.8.2022]

SAŽETAK

Razvojem mobilnih tehnologija omogućeno je da ljudi budu pravovremeno obaviješteni o određenim informacijama u svijetu korištenjem mobilnih uređaja. Cilj rada bio je implementacija mobilne aplikacije koja korisnicima omogućava praćenje informacija o virusu koji je znatno utjecao na svjetsku ekonomiju. Na početku rada opisani su već postojeće aplikacije koje nude mogućnost praćenja informacija o koronavirusu. Zatim je opisan programski jezik Swift i programski okviri u fazi razvoja Swift-UI i Combine koji su bili potrebni za uspješno stvaranje mobilne aplikacije. Definirana je modularna arhitektura programskog koda i organizacija datoteka u projektu. Na kraju je detaljnije opisana implementacija koda te su prikazani svi zasloni koji mogu biti prikazani na mobilnom uređaju.

Commented [JB13]: Dodajte ključne riječi i keywords

Ključne riječi: iOS, koronavirus, mobilna aplikacija, pametni mobilni uređaji, Swift

ABSTRACT

The development of mobile technologies has made it possible for people to be informed in a timely manner about certain information in the use of mobile devices in the world. The goal of the work was the implementation of a mobile application that allows users to track information about the virus that has significantly affected the world economy. At the beginning of the work, already existing applications are described that offer the possibility of monitoring information about the coronavirus. Then the programming language Swift and the programming frameworks in the development phase Swift-UI and Combine, which were necessary for the successful creation of mobile applications, were described. The modular architecture of the program code and the organization of files in the project are defined. At the end, the implementation of the code is described in more detail, and all the screens that can be displayed on a mobile device are shown.

Keywords: iOS, coronavirus, mobile application, smartphones, Swift

ŽIVOTOPIS

Josip Juhasz rođen je 31.07.1998. godine u Đakovu. Živi u Selcima Đakovačkim na adresi Bana Josipa Jelačića 97a. Osnovnoškolsko obrazovanje započinje 2005. godine u OŠ Đakovački Selci u Selcima Đakovačkim. Nakon osnovnoškolskog obrazovanja, 2013. godine upisuje Gimnaziju Antuna Gustava Matoša u Đakovu. Nakon srednjoškolskog obrazovanja upisuje Fakultet elektrotehnike, računarstva i informacijskih tehnologija, sveučilišni studij, smjer Računarstvo. Nakon završenog preddiplomskog studija, upisuje diplomski studij Informacijske i podatkovne znanosti na istom fakultetu. Posjeduje znanje u govoru, čitanju i pisanju engleskog jezika, te vozačku dozvolu B kategorije. Osim navedenog posjeduje i osnovno znanje programskih alata, odnosno tehnologija: iOS, Swift, HTML, CSS, C, C++, C# i JAVA te Microsoft Office alatima.

Commented [JB14]: Ovbostrano poravnanje

Commented [JB15]: Ja sam proveo dio djetinjstva u Selcima 😉

Commented [JB16]: ?