Aplikacija za provjeru zaraženosti s COVID19 na osnovu simptoma

Delić, Igor

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni studij

APLIKACIJA ZA PROVJERU ZARAŽENOSTI S COVID19 NA OSNOVU SIMPTOMA

Završni rad

Igor Delić

Osijek, 2022.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

Obrazac Z1P - Obrazac za ocjenu završnog rada na preddiplomskom sveučilišnom studiju

Osijek, 14.09.2022.

Odboru za završne i diplomske ispite

Prijedlog ocjene završnog rada na preddiplomskom sveučilišnom studiju

Ime i prezime Pristupnika:	Igor Delić
Studij, smjer:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. Pristupnika, godina upisa:	R 4449, 22.07.2019.
OIB Pristupnika:	82647560152
Mentor:	Izv. prof. dr. sc. Alfonzo Baumgartner
Sumentor:	,
Sumentor iz tvrtke:	
Naslov završnog rada:	Aplikacija za provjeru zaraženosti s COVID19 na osnovu simptoma
Znanstvena grana rada:	**Obradba informacija (zn. polje računarstvo)**
Zadatak završnog rad:	[Rezervirano: Igor Delić] Napraviti mobilnu aplikaciju koja će na temelju unosa simptoma od strane korisnika procijeniti i dati odgovor koliko se ti simptomi preklapaju sa zaraženosti COVID19. Također korisnik treba unijeti i broj dana koliko dugo osjeća simptome, trenutnu temperaturu, najvišu temperaturu i slično. Na temelju rezultata, aplikacija treba dodatno ispisati odgovarajuće prijedloge korisniku što treba činiti.
Prijedlog ocjene završnog rada:	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene od strane mentora:	14.09.2022.
Datum potvrde ocjene od strane Odbora:	21.09.2022.

Potvrda mentora o predaji konačne verzije rada:	*Mentor elektronički potpisao predaju konačne verzije.*
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 21.09.2022.

Ime i prezime studenta:	Igor Delić
Studij:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	R 4449, 22.07.2019.
Turnitin podudaranje [%]:	9

Ovom izjavom izjavljujem da je rad pod nazivom**: Aplikacija za provjeru zaraženosti s COVID19 na osnovu simptoma**

izrađen pod vodstvom mentora Izv. prof. dr. sc. Alfonzo Baumgartner

i sumentora ,

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD

Zadatak završnog rada je bila izrada mobilne aplikacije pod nazivom „Aplikacija za provjeru zaraženosti s COVID19 na osnovu simptoma“. Aplikacija na temelju unosa simptoma od strane korisnika procijeniti radi li se o bolesti Covid19 ili je riječ o sličnim oblicima bolesti poput prehlade ili gripe te na temelju rezultata dodatno pruža odgovarajuće prijedloge korisniku što treba učiniti. Svaki korisnik također ima mogućnost unosa trenutne temperature, najviše temperature i broja dana koliko osjeća simptome te se podaci unose u bazu podataka. Korisnik pristupa svojim podacima pomoću prethodno unešenog korisničkog imena.

U stavci „Programski jezik i alati“ će se proći kroz korišteni programski jezik Kotlin, razvojno okruženje Android Studio u kojem je aplikacija realizirana i Firebase. U stavci „Izrada aplikacije“ će se proći kroz strukturu, način izrada i princip rada same aplikacije, a na samom kraju će se proći kroz zaključak i sažetak rada.

1.1. Zadatak završnog rada

Aplikacija korisniku pruža uslugu provjere zaraženosti bolešću uzrokovanom korona virusom ili sličnim bolestima te ga savjetuje o daljnjnim postupcima. Također pruža evidenciju podataka svakog korisnika aplikacije.

2. PROGRAMSKI JEZIK I ALATI

U ovom poglavlju će se proći kroz sve korištene alate pri izradi aplikacije kao i korišteni programski jezik u kojem je aplikacija napravljena.

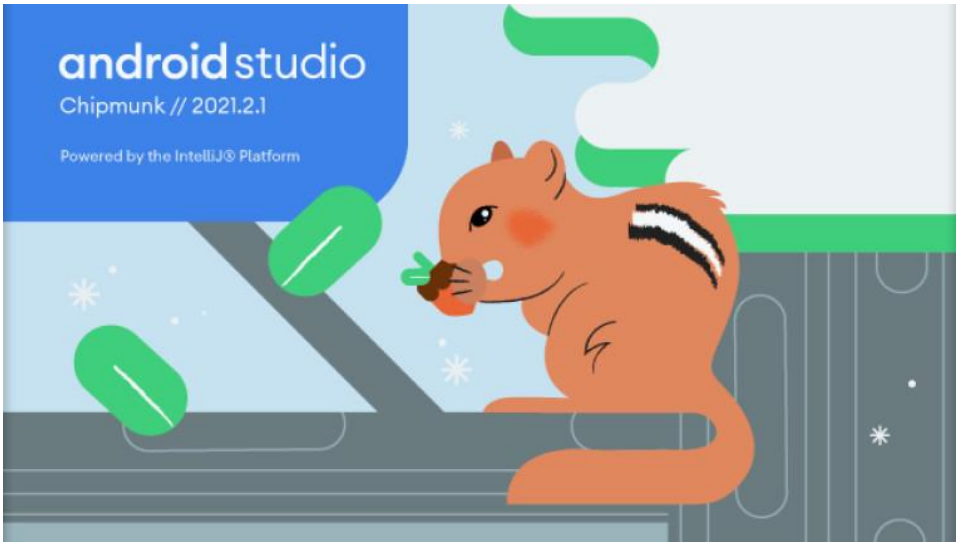
2.1. Kotlin

Kotlin je višeplatformski programski jezik opće dizajniran da bude u potpunosti interoperabilan s Javom, a Java Virtual Machine verzija standardne biblioteke Kotlin oslanja se na biblioteke Java klasa, ali s čišćom sintaksom. Troškove razvoja jezika pokriva JetBrains, a Kotlin Foundation štiti zaštitni znak Kotlin (Slika 2.1.). Dana 7. svibnja 2019. Google je objavio da je programski jezik Kotlin sada jezik izbora za njegove programere aplikacija za Android. Od izdanja Android Studija 3.0 u listopadu 2017., Kotlin je ugrađen kao alternativa Java prevoditelju.

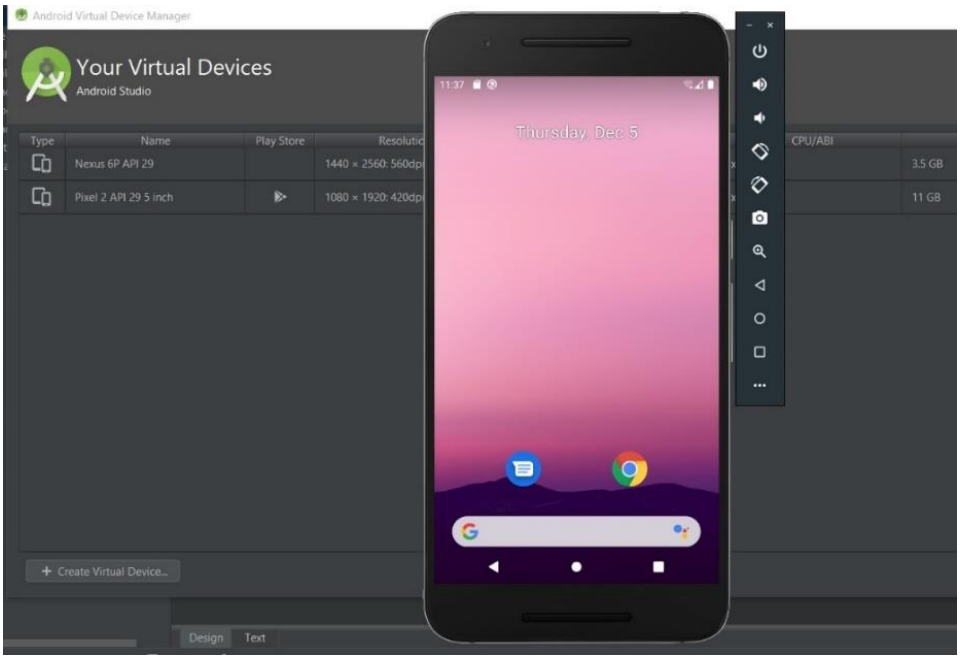
Slika 2.1. *Prikaz zaštitnog znaka Kotlin.*

2.2. Android Studio

Android Studio je službeni IDE za operativni sustav Android, izgrađen na JetBrainsovom IntelliJ IDEA softveru i dizajniran za Android razvoj. Android Studio je najavljen na Google I/O 16. svibnja 2013., a prva stabilna verzija objavljena je u prosincu 2014., počevši od verzije 1.0.. Dostupno za preuzimanje na operativnim sustavima Windows, macOS i Linux. Android Studio podržava programske jezike Java i C++, dok Android Studio 3.0 ili noviji podržava Kotlin. Također, sadrži Android Emulator(Slika 2.3.) koji simulira razne android uređaje i znatno olakšava testiranje aplikacije. Nakon što kompajlirate svoju aplikaciju, možete je objaviti u trgovini Google Play.



Slika 2.2. *Najnovija verzija Android Studio-a „Chipmunk“.*



Slika 2.3. *Prikaz Android Studio Emulator-a.*

2.3. Firebase

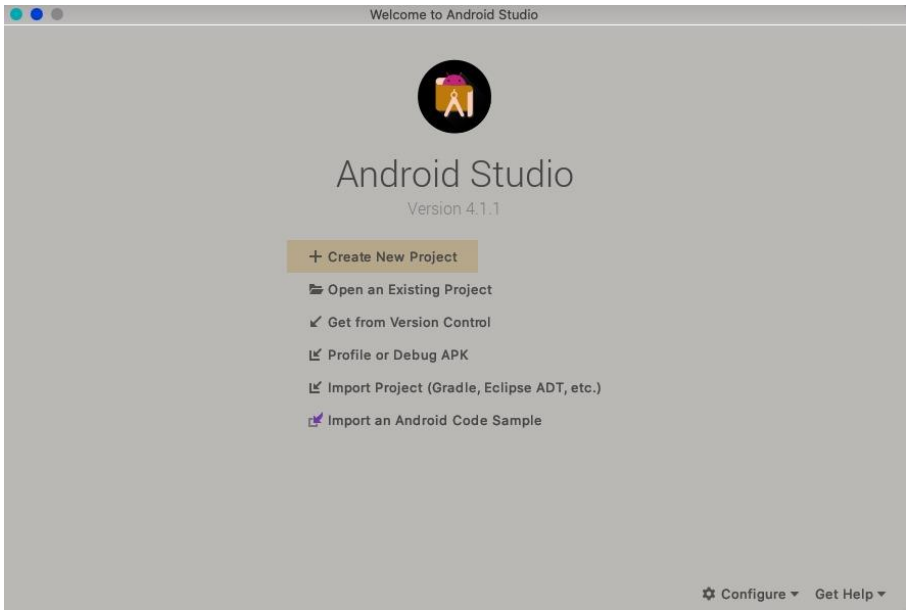
Firebase je platforma za izradu mobilnih aplikacija. Razvio iz Envolvea, startupa koji su osnovali James Tamplin i Andrew Lee 2011. godine. Envolve pruža programerima API za integraciju funkcije live chata na svoje web stranice. Nakon što je chat usluga puštena u promet, Tamplin i Lee otkrili su da se koristi za prosljeđivanje podataka aplikacije osim poruka chata. Envolve se koristio za sinkronizaciju podataka aplikacije, npr. stanje igre, u stvarnom vremenu između korisnika. Chat sustav je odvojen od arhitekture u stvarnom vremenu koja ga podržava. Pokrenuli su Firebase kao neovisnu tvrtku 2011. i izašli na burzu 2012. Godine. Platformu je kupio Google 2014. i sada je njihov glavni proizvod za razvoj aplikacija.

Slika 2.4. *Prikaz novog Firebase loga.*

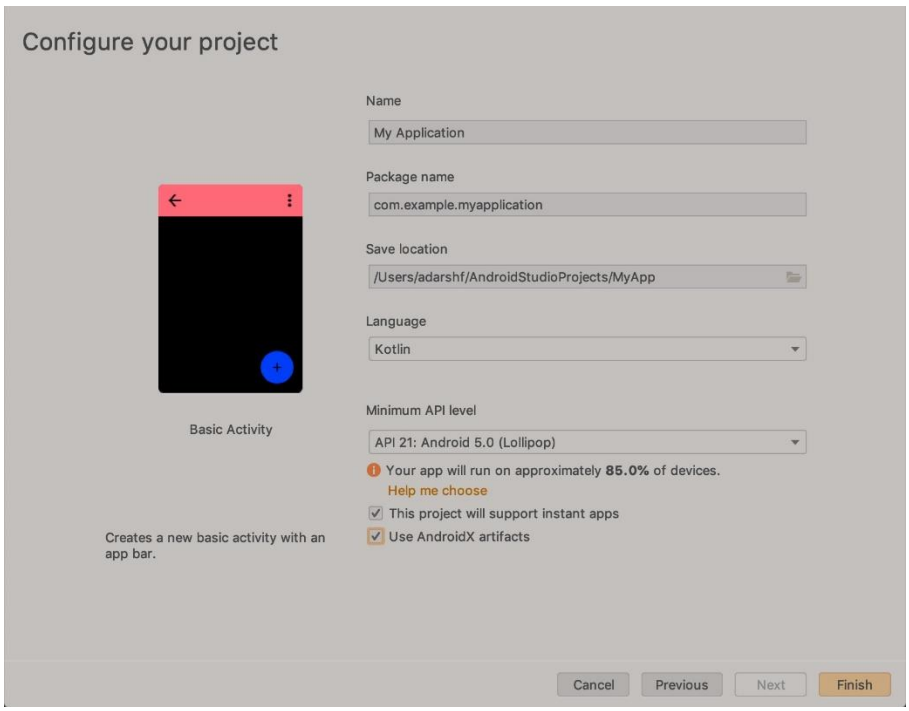
Jedan od prvih njihovih proizvoda je Firebase Realtime Database, poseban API koji sprema podatke na Firebaseov oblak. Upravo je Realtime Database korišten u izradi aplikacije za pohranu i dohvaćanje korisničkih podataka .

3. IZRADA APLIKACIJE

Prije početka izrade aplikacije bilo je potrebno detaljno proučiti i istražiti područje rada, smisliti dizajn i način rada aplikacije. Aplikacija se sastoji od 8 aktivnosti i klase korisnika. Android aplikacije podijeljene su na aktivnosti, a svaka aktivnost odnosi se na jedan ekran. Pri izradi novog projekta stvorena je prva aktivnost koja je ujedno i početni zaslon aplikacije(Slika 3.1.).



Slika 3.1. *Prikaz kreiranja novog projekta u Android Studio-u.*



Slika 3.2. *Prikaz kreiranja nove aktivnosti*.

3.1. Početni zaslon

Slike, izgled i drugi podaci izdvojeni su od ostatka koda, a nazvaju se resursima i spremljeni su u posebnu mapu. Izgled aplikacije se određuje uz pomoće layouta, datoteka napisanih XML opisnim jezikom. Svaki layout mora sadržavati korijenski element, koji mora biti View ili ViewGroup objekt. U korijenski se element zatim dodaju drugi elementi i stvara se svojevrsna hijerarhija. View je osnovni sastavni blok korisničkog sučelja, a predstavlja mali pravokutni okvir. Pri pokretanju aplikacije otvara se početni zaslon, odnosno aktivnost MainActivity. Ova aktivnost sadrži ImageView, TextView i dva Button-a(Slika 3.3.).

Slika 3.3. *Prikaz XML početnog zaslona*.

TextView je klasa za prikaz tekstualnog sadržaja koji korisnik ne može mijenjati ni uređivati. Da bi se dodao TextView unutar layouta potrebno je unutar XML datoteke dodati tag <TextView>. Pomoću svojstava layout_width i layout_height određuju se visina i širina elementa, a svojstvo text određuje sadržaj koji se nalazi u elementu. Sa svojstvom layout_constraint određuje se položaj elementa u odnosu na druge elemente(Slika 3.4.).

```
<TextView
    android:id="@+id/textView2"
    android:layout_width="340dp"
    android:layout_height="122dp"
    android:layout_marginTop="20dp"
    android:layout_marginBottom="50dp"
    android:fontFamily="@font/architects_daughter"
    android:text="Je li i vama zdravlje na prvom mjestu? O zdravlju se brine dok si još zdrav, no mi smo tu da vam pomognemo !"
    android:textColor="#FFFFFF"
    android:textSize="20dp"
    app:layout_constraintBottom_toTopOf="@+id/simptomibtn"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.491"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/imageView5" />
```

Slika 3.4. *Prikaz TextView koda*.

Klasa Button predstavlja gumb na čiji pritisak možemo obaviti neki posao ili radnju. Da bi se Button dodao u layouta potrebno je u XML datoteku dodati tag <Button>. Svojstvima layout_width i layout_height dimenzije visine i širine elementa postavljaju se da budu jednake veličini sadržaja koji se nalazi unutar njega(Slika 3.5.).

```
<Button
    android:id="@+id/simptomibtn"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="20dp"
    android:layout_marginBottom="5dp"
    android:backgroundTint="@color/red"
    android:text="Simptomi"
    android:textColor="#FFFFFF"
    android:textSize="20sp"
    app:layout_constraintBottom_toTopOf="@+id/databtn"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/textView2" />
```

Slika 3.5. *Prikz Button koda*.

Klasa ImageView prikaz slike na zaslonu. Da bi se dodao unutar layouta potrebno je u XML datoteku dodati tag <ImageView>, a sva korištena svojstva su prethodno opisana i objašnjena(Slika 3.6.).

```
26      <ImageView
27          android:id="@+id/imageView5"
28          android:layout_width="373dp"
29          android:layout_height="288dp"
30          android:layout_marginTop="10dp"
31          android:layout_marginBottom="50dp"
32          app:layout_constraintBottom_toTopOf="@+id/textView2"
33          app:layout_constraintEnd_toEndOf="parent"
34          app:layout_constraintStart_toStartOf="parent"
35          app:layout_constraintTop_toTopOf="parent"
36          app:srcCompat="@drawable/image" />
```

3.6. *Prikaz ImageView koda*.

Glavna funkcionalnost MainActivity-a je da predstavlja početni izbornik korisniku. Aplikacija se sastoji od više aktivnosti potrebno je omogućiti promjenu od jedne do druge aktivnosti, a to omogućava Intent. Korištenjem Intenta moguće je pokrenuti bilo koju aktivnost, a razlikujemo implicitne i eksplicitne Intente. Eksplicitni definiraju komponentu u samom Intentu, dok implicitni traže od sustava da sam procjeni komponente. Intent također omogućuje i komunikaciju među drugim komponentama aplikacije. Korisnik pritiskom na određeni Button prelazi na drugu, odgovarajuću aktivnost. Pritiskom na Button „Simptomi“ prelazi u aktivnost Simptomi, a pritiskom na Button „Moji podaci“ prelazi u aktivnost Ocitaj(Slika 3.7.).

MainActivity.kt

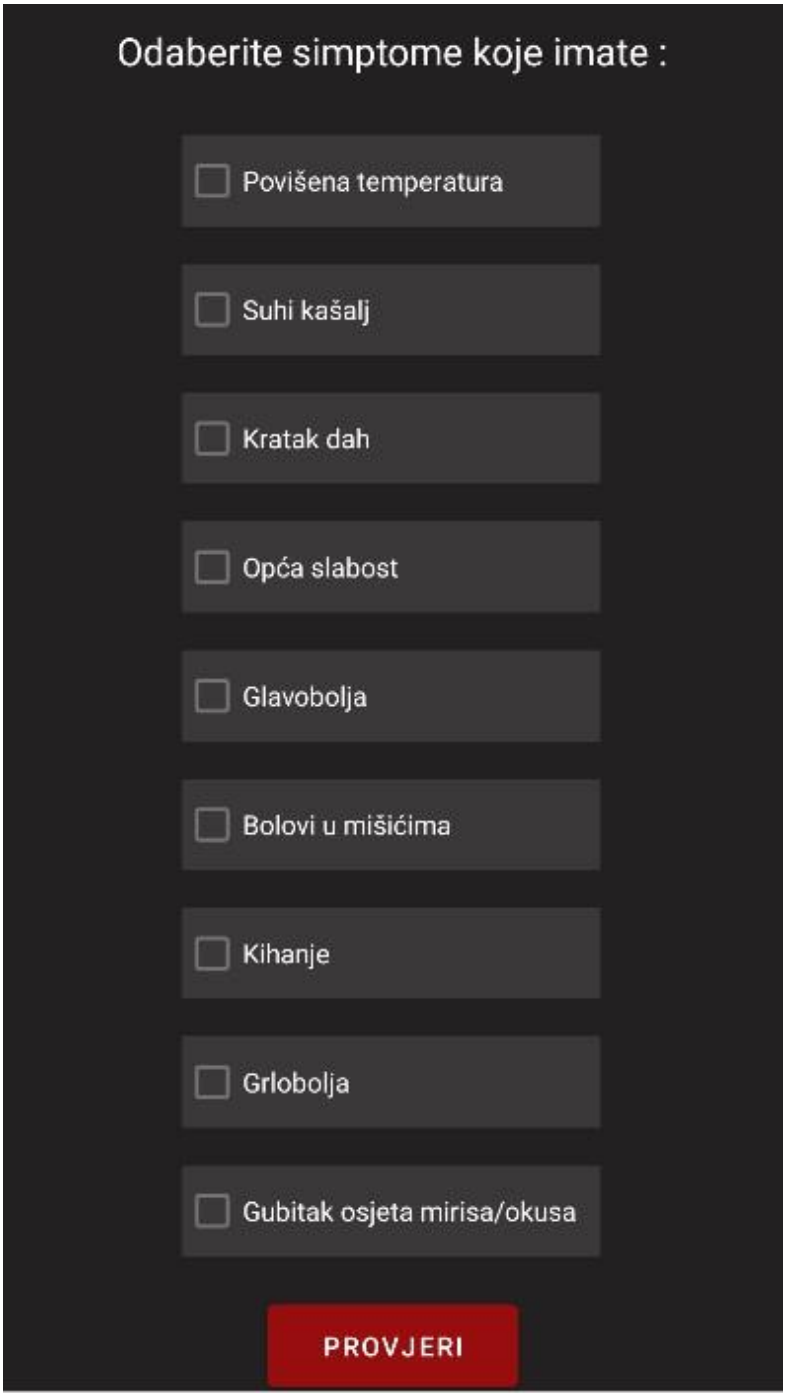
```
1   package com.example.applicationtocheckforcovid19infectionbasedonsymptoms
2
3   import ...
9
10  class MainActivity() : AppCompatActivity() {
11
12      override fun onCreate(savedInstanceState: Bundle?) {
13          super.onCreate(savedInstanceState)
14          setContentView(R.layout.activity_main)
15
16          val secndActivButton = findViewById<Button>(R.id.simptomibtn)
17          secndActivButton.setOnClickListener {  it: View!
18              val Intetn = Intent( packageContext: this,Simptomi::class.java)
19              startActivity(Intetn)
20          }
21
22          val thirdActivButton = findViewById<Button>(R.id.databtn)
23          thirdActivButton.setOnClickListener {  it: View!
24              val Intent2 = Intent( packageContext: this,Ocitaj::class.java)
25              startActivity(Intent2)
26          }
27      }
28  }
```

Slika 3.7. *Prikaz MainActivity koda.*

Korišten je eksplicitni Intent, a za pokretanje aktivnosti potrebno je pozvati startActivity metodu kojoj se kao argument prosljeđuje Intent objekt.

3.2. Simptomi

Korisnik pritiskom na Button „Simptomi“ u MainActivity-u prelazi u ovu aktivnost. Aktivnost se sastoji od TextView-a, Button-a i CheckBox-ova. CheckBox je posebna vrsta Button-a koji ima dvije opcije, označeno ili neoznačeno. Dosta često se koristi jer je vrlo praktičan i pruža korisniku opciju odgovora na pitanje s višestrukim točnim odgovorima ili odabira više različitih stavki, a našao je primjenu i u ovoj aplikaciji.



Slika 3.8. *Prikaz XML Simptomi.*

Da bi se CheckBox dodao u layout potrebno je u XML dodati tag <CheckBox>. Svojstvima layout_width i layout_height definiraju se dimenzije visine i širine CheckBox elementa i svi CheckBox-ovi u ovoj aktivnosti su istih dimenzija(Slika 3.9.).

```
25      <CheckBox
26          android:id="@+id/checkBox"
27          android:layout_width="220dp"
28          android:layout_height="48dp"
29          android:layout_marginTop="10dp"
30          android:layout_marginBottom="10dp"
31          android:textColor="#ffffff"
32          android:text="Povišena temperatura"
33          android:background="@color/box"
34          app:layout_constraintBottom_toTopOf="@+id/checkBox2"
35          app:layout_constraintEnd_toEndOf="parent"
36          app:layout_constraintHorizontal_bias="0.492"
37          app:layout_constraintStart_toStartOf="parent"
38          app:layout_constraintTop_toBottomOf="@+id/textView" />
```

Slika 3.9. *Prikaz CheckBox koda.*

Također, u ovoj aktivnost je dodana alatna traka u slučaju da se korisnik odlučni za povratak na početnu aktivnost. U AndroidManifest.xml datoteku je MainActivity postavljen kao roditelj ovoj aktivnosti i u sami kod aktivnosti je dodana linija za podršku alatne trake u aplikaciji(Slika 3.9.).

```
16      <activity
17          android:name=".Ocitaj"
18          android:exported="false"
19          android:parentActivityName=".MainActivity"/>
```

Slika 3.9. Postavljanje roditeljske aktivnosti.

```
17      supportActionBar!!.setDefaultDisplayHomeAsUpEnabled(true)
```

Slika 3.9. *Podrška za alatnu traku.*

U ovoj aktivnosti korisnik bira simptome koje ima označavajući ih u CheckBox-u. Nakon što označi sve simptome koje osjeća potrebno je kliknuti Button „Provjeri“. Na temelju označenih simptoma aplikacija vraća korisniku povratnu informaciju radi li se o bolesti Covid19, gripi ili samo običnoj prehladi. Razlike u simptomima kod ove tri bolesti su vrlo male i često zbunjujuće. Prilikom istraživanja napravljene su liste sa svim kombinacijama simptoma koje ukazuju na određenu vrstu bolesti. Kada korisnik dolazi u ovu aktivnost radi se korisnička prazna lista. Nakon što je korisnik označio simptome, pritiskom na Button označeni simptomi se dodaju u korisničku listu. Zatim se korisnička lista uspoređuje s listama simptoma provedenih u istraživanju i na temelju toga korisnik saznaje o kojoj bolesti se radi(Slika 3.11. - 3.14.).

Simptomi.kt

```
package com.example.applicationtocheckforcovid19infectionbasedonsymptoms

import ...

class Simptomi() : AppCompatActivity() {

    @SuppressLint("RestrictedApi")
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_simptomi)

        supportActionBar!!.setDefaultDisplayHomeAsUpEnabled(true)
        val chk = findViewById<CheckBox>(R.id.checkBox)
        val chk2 = findViewById<CheckBox>(R.id.checkBox2)
        val chk3 = findViewById<CheckBox>(R.id.checkBox3)
        val chk4 = findViewById<CheckBox>(R.id.checkBox4)
        val chk5 = findViewById<CheckBox>(R.id.checkBox5)
        val chk6 = findViewById<CheckBox>(R.id.checkBox6)
        val chk7 = findViewById<CheckBox>(R.id.checkBox7)
        val chk8 = findViewById<CheckBox>(R.id.checkBox8)
        val chk9 = findViewById<CheckBox>(R.id.checkBox9)

        val grip1 = mutableListOf<String>("Temp","Dah","Slab","Glav","Grl","Bol")
        val grip2 = mutableListOf<String>("Temp","Dah","Slab","Glav","Bol")
        val grip3 = mutableListOf<String>("Temp","Dah","Slab","Glav","Grl")
        val grip4 = mutableListOf<String>("Temp","Slab","Glav","Grl","Bol")
        val grip5 = mutableListOf<String>("Temp","Glav","Grl","Bol")
        val grip6 = mutableListOf<String>("Temp","Glav","Bol")
        val grip7 = mutableListOf<String>("Temp","Dah","Slab","Glav")
        val grip8 = mutableListOf<String>("Temp","Dah","Slab","Bol")
        val grip9 = mutableListOf<String>("Temp","Slab","Bol")
        val grip10 = mutableListOf<String>("Temp","Dah","Slab")
        val grip11 = mutableListOf<String>("Temp","Dah","Bol")
        val grip12 = mutableListOf<String>("Temp","Bol")

        val preh1 = mutableListOf<String>("Slab","Kih")
        val preh2 = mutableListOf<String>("Kih","Grl")
        val preh3 = mutableListOf<String>("Slab","Grl")
        val preh4 = mutableListOf<String>("Kih")
        val preh5 = mutableListOf<String>("Grl")
```

Slika 3.11. *Prikaz koda Simptomi.*

```
49      val cov1 = mutableListOf<String>("Temp","Kas","Dah","Slab","Bol","Grl","Glav","Osj")
50      val cov2 = mutableListOf<String>("Temp","Kas","Dah","Slab","Bol","Grl","Glav")
51      val cov3 = mutableListOf<String>("Temp","Kas","Dah","Slab","Bol","Grl","Osj")
52      val cov4 = mutableListOf<String>("Temp","Kas","Dah","Slab","Bol","Osj")
53      val cov5 = mutableListOf<String>("Temp","Kas","Dah","Slab","Osj")
54      val cov6 = mutableListOf<String>("Temp","Kas","Dah","Bol","Osj")
55      val cov7 = mutableListOf<String>("Temp","Kas","Slab","Bol","Osj")
56      val cov8 = mutableListOf<String>("Temp","Dah","Slab","Bol","Osj")
57      val cov9 = mutableListOf<String>("Temp","Slab","Bol","Osj")
58      val cov10 = mutableListOf<String>("Temp","Slab","Bol","Osj")
59      val cov11 = mutableListOf<String>("Temp","Dah","Bol","Osj")
60      val cov12 = mutableListOf<String>("Temp","Kas","Dah","Osj")
61      val cov13 = mutableListOf<String>("Temp","Dah","Slab","Osj")
62      val cov14 = mutableListOf<String>("Temp","Dah","Slab","Osj")
```

Slika 3.12. *Prikaz koda Simptomi.*

```
64      val symps = mutableListOf<String>()
65      symps.clear()
66      val checkButton = findViewById<Button>(R.id.chcButton)
67      checkButton.setOnClickListener {  it: View!
68          symps.clear()
69          if (chk.isChecked){
70              symps.add("Temp")
71          }
72          if (chk2.isChecked){
73              symps.add("Kas")
74          }
75          if (chk3.isChecked){
76              symps.add("Dah")
77          }
78          if (chk4.isChecked){
79              symps.add("Slab")
80          }
81          if (chk5.isChecked){
82              symps.add("Glav")
83          }
84          if (chk6.isChecked){
85              symps.add("Bol")
86          }
87          if (chk7.isChecked){
88              symps.add("Kih")
89          }
90          if (chk8.isChecked){
91              symps.add("Grl")
92          }
93          if (chk9.isChecked){
94              symps.add("Osj")
95          }
```

Slika 3.13. *Prikaz koda Simptomi.*

Simptomi.kt

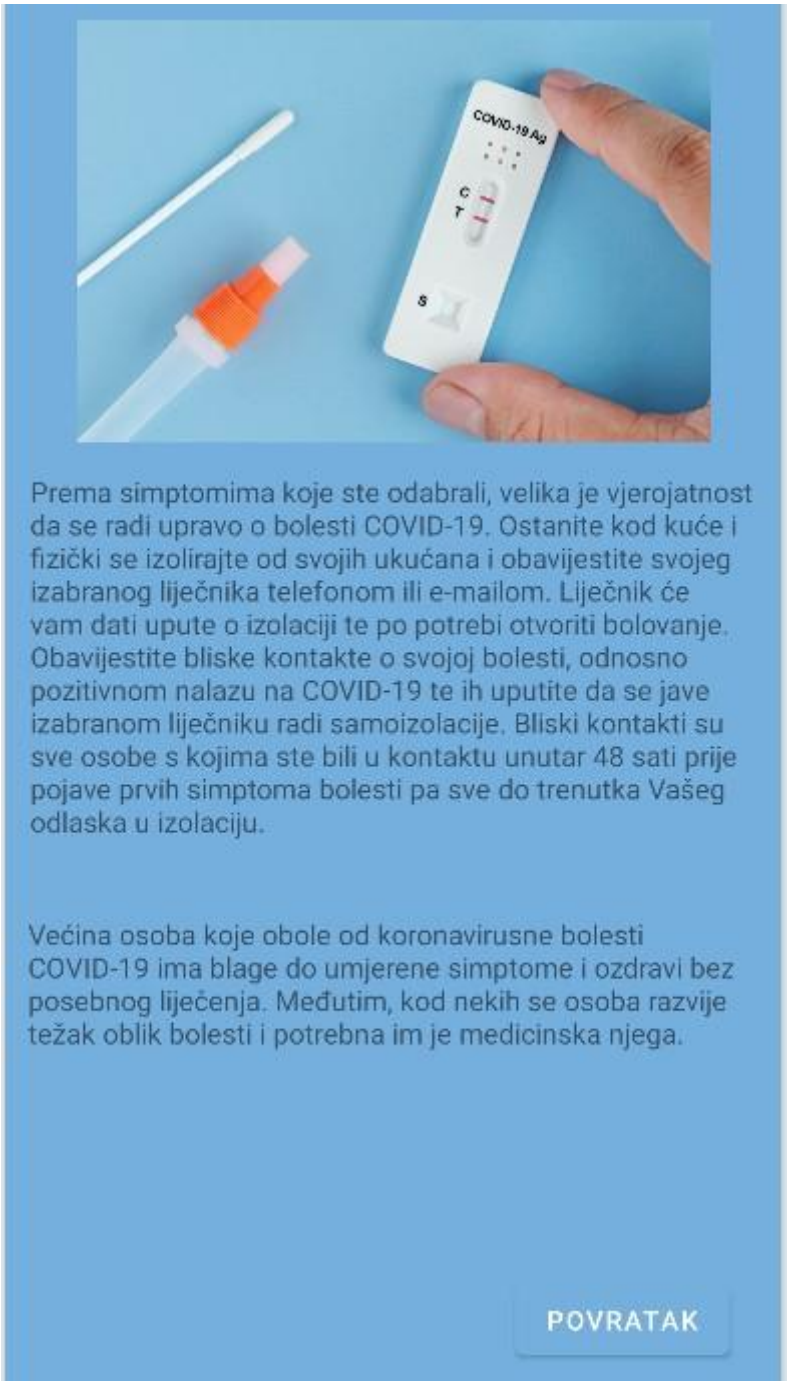
```
98      if(symps.containsAll(cov1)){
99          val Intent = Intent( packageContext: this,Covid::class.java)
100         startActivity(Intent)
101         symps.clear()
102     }
103     else if(symps.containsAll(cov2)){
104         val Intent = Intent( packageContext: this,Covid::class.java)
105         startActivity(Intent)
106         symps.clear()
107     }
108     else if(symps.containsAll(cov3)){
109         val Intent = Intent( packageContext: this,Covid::class.java)
110         startActivity(Intent)
111         symps.clear()
112     }
113     else if(symps.containsAll(cov4)){
114         val Intent = Intent( packageContext: this,Covid::class.java)
115         startActivity(Intent)
116         symps.clear()
117     }
118     else if(symps.containsAll(cov5)){
119         val Intent = Intent( packageContext: this,Covid::class.java)
120         startActivity(Intent)
121         symps.clear()
122     }
123     else if(symps.containsAll(cov6)){
124         val Intent = Intent( packageContext: this,Covid::class.java)
125         startActivity(Intent)
126         symps.clear()
127     }
128     else if(symps.containsAll(cov7)){
129         val Intent = Intent( packageContext: this,Covid::class.java)
130         startActivity(Intent)
131         symps.clear()
132     }
133     else if(symps.containsAll(cov8)){
134         val Intent = Intent( packageContext: this,Covid::class.java)
135         startActivity(Intent)
136         symps.clear()
```

Slika 3.14. *Prikaz koda Simptomi.*

Pomoću metode containsAll() se provjerava u kojoj listi simptoma se nalaze korisnikovi simptomi. Ako se korisnikovi simptomi ne poklapaju s niti jednom od lista iz istraživanja aplikacija savjetuje korisnika da još jednom provjeri unos simptoma ili da pričeka moguću pojavu novih simptoma.

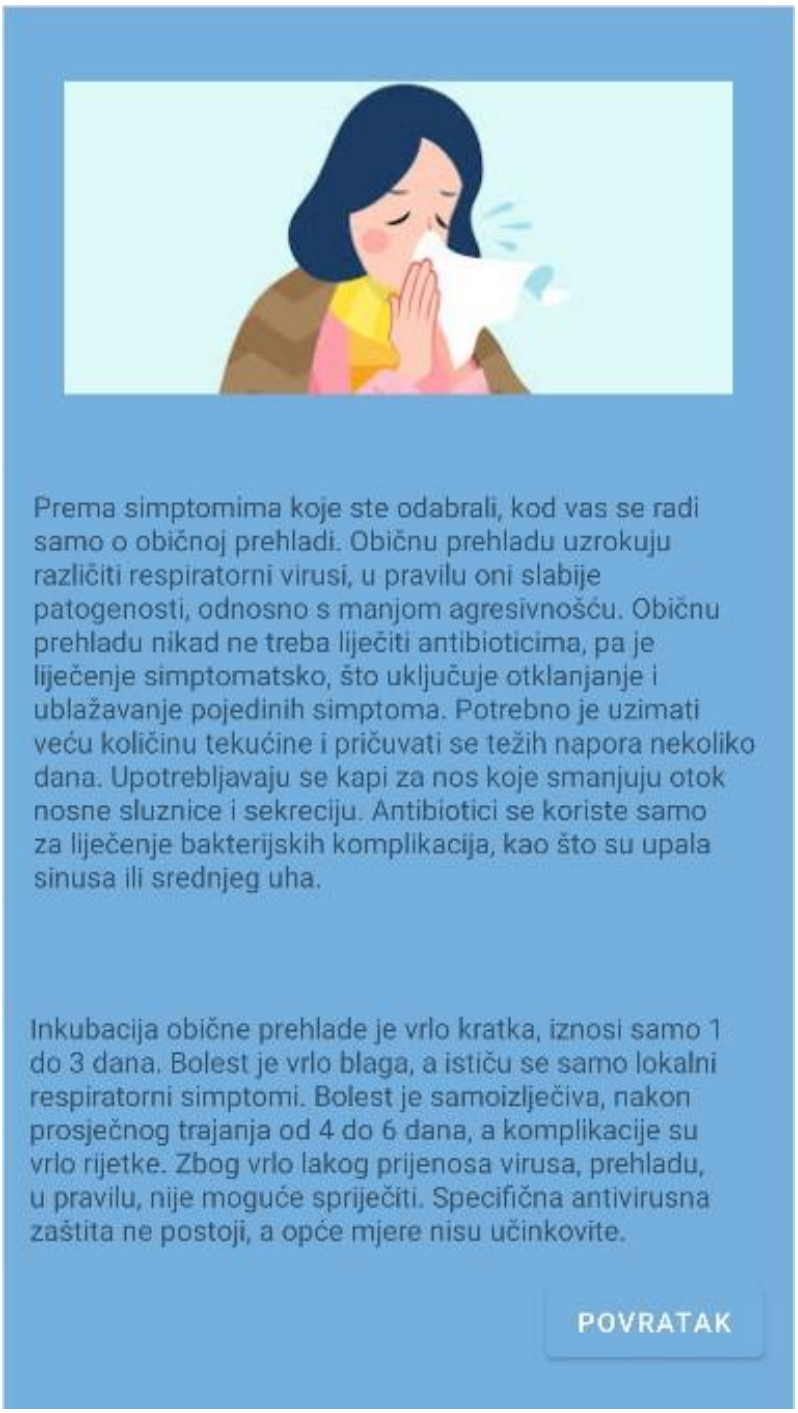
3.3. Rezultati provjere

Nakon što aplikacija nađe odgovarajuću listu simptoma i uz njih pripadnu bolest, pomoću Intenta prelazi na odgovarajuću aktivnost ovisno o odabranoj vrsti bolesti. Svaka aktivnost sadrži TextView, ImageView i Button „Povratak“ koji pomoću Intenta vraća korisnika na početni zaslon. Svaka od aktivnosti sadrži informacije o vrsti bolesti o kojoj se radi te kako bi korisnik trebao dalje postupati(Slika 3.15. - 3.18.).



Slika 3.15.. *Prikaz aktivnosti Covid.*

Slika 3.16.. *Prikaz aktivnosti Gripa.*



Slika 3.17.. *Prikaz aktivnosti Prehlada.*

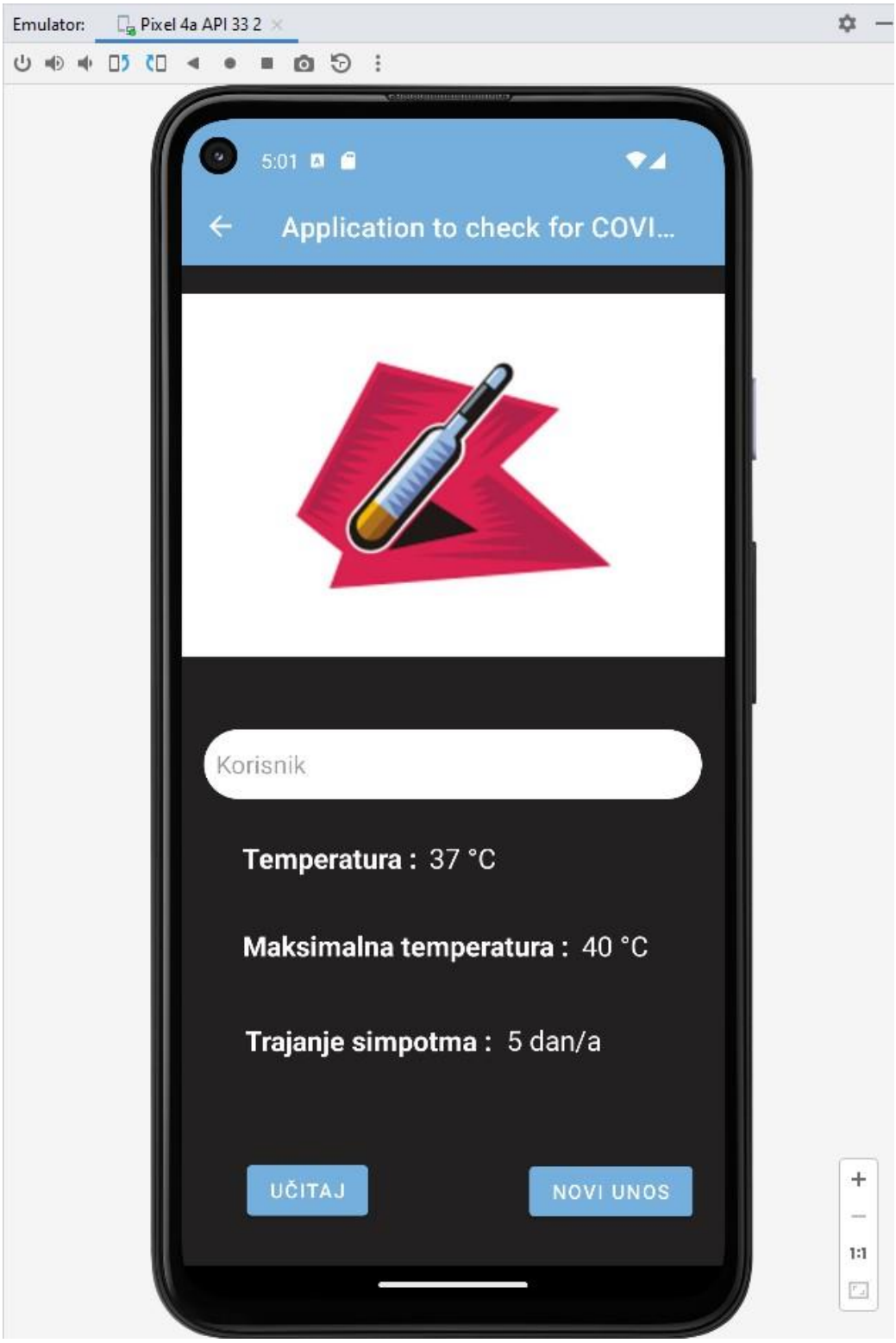
Slika 3.18.. *Prikaz aktivnosti Neodređeno.*

3.4. Podaci korisnika

Kada korisnik na početnom zaslonu, u aktivnosti MainActivity pristisne Button „Moji podaci“, prelazi u aktivnosti Ocitaj. Aktivnost se sastoji od ImageView-a, dva Button-a i šest TextView-a. Također je dodan i EditText kako bi korisnik mogao unijeti korisničko ime. To je klasa koja predstavlja UI kontrolu za unos niza znakova. Da bi se dodao EditText unutar layouta potrebno je unutar XML datoteke dodati oznaku <EditText>. Svojstvo text određuje sadržaj koji se nalazi unutar Viewa prije no što korisnik klikne na njega. Svojstvom inputType se određuju pravila unosa znakova odnosno koja vrsta tipkovnice će se prikazati korisniku kada se klikne na EditText. Pomoću svojstva hint korisniku dajemo do znanja kakav se unos očekuje. Također je dodana alatna traka s opcijom za povratak u MainActivity(Slika 3.19.).

Slika 3.19.. *Prikaz XML Ocitaj*.

Nakon što korisnik unese korisničko ime može pritisnuti Button „Učitaj“ i aplikacija će iz Firebase-a povući korisnikove podatke i prikazati ih na zaslonu(Slika 3.20.).



Slika 3.20.. *Prikaz XML Ocitaj nakon učitavanja podataka.*

Ocitaj.kt

```kotlin
class Ocitaj : AppCompatActivity() {
    private lateinit var binding: ActivityOcitajBinding
    private lateinit var database: DatabaseReference

    @SuppressLint("RestrictedApi")
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_ocitaj)
        supportActionBar!!.setDefaultDisplayHomeAsUpEnabled(true)
        binding = ActivityOcitajBinding.inflate(layoutInflater)
        setContentView(binding.root)
        val swapBtn = findViewById<Button>(R.id.unsBtn)
        swapBtn.setOnClickListener { it: View!
            val Intent = Intent( packageContext: this, Podaci::class.java)
            startActivity(Intent)
        }

        binding.loadBtn.setOnClickListener { it: View!
            val userName : String = binding.etUsername.text.toString()
            if(userName.isNotEmpty()){
                readData(userName)
            }else{
                Toast.makeText( context: this, text: "Unesite korisnicko ime",Toast.LENGTH_SHORT).show()
            }
        }

    }

```

Slika 3.21.. *Prikaz koda Ocitaj*.

```kotlin
    private fun readData(userName: String) {
        database = FirebaseDatabase.getInstance().getReference( path: "Users")
        database.child(userName).get().addOnSuccessListener { it: DataSnapshot!
            if(it.exists()){

                val trentemp = it.child( path: "trenTemp").value
                val maxtemp = it.child( path: "maxTemp").value
                val sypbegin = it.child( path: "sympBegin").value
                Toast.makeText( context: this, text: "Uspjesno ucitano",Toast.LENGTH_SHORT).show()
                binding.etUsername.text.clear()
                binding.tvTren.text = trentemp.toString() + " °C"
                binding.tvMax.text = maxtemp.toString() + " °C"
                binding.tvDani.text = sypbegin.toString() + " dan/a"


            }else{
                Toast.makeText( context: this, text: "Korisnik ne postoji",Toast.LENGTH_SHORT).show()
            }
        }.addOnFailureListener { it: Exception
            Toast.makeText( context: this, text: "ERROR",Toast.LENGTH_SHORT).show()
        }
    }

}
```

Slika 3.22.. *Prikaz koda Ocitaj*.

U slučaju da korisnik do sada još nije unosio svoje podatke u aplikaciju, pritiskom na Button „Novi unos“, prelazi u aktivnost Podaci. Aktivnost se sastoji od četiri EditText-a, ImageView-a i Buttona na čiji pritisak se unešeni podaci unose u bazu podataka(Slika 3.23.). Da bi to bilo moguće, bilo je potrebno napraviti posebnu klasu User koja predstavlja svakog unešenog korisnika u aplikaciju i aplikaciju povezati s Firebase-om.

Slika 3.23.. *Prikaz XML Podaci.*

Podaci.kt

```kotlin
import ...

class Podaci : AppCompatActivity() {
    @SuppressLint("RestrictedApi")
    private lateinit var binding: ActivityPodaciBinding
    private lateinit var database: DatabaseReference
    @SuppressLint("RestrictedApi")
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityPodaciBinding.inflate(layoutInflater)
        setContentView(binding.root)
        binding.button.setOnClickListener {  it: View!
            val trenTemp = binding.unos1.text.toString()
            val maxTemp = binding.unos2.text.toString()
            val sympBegin = binding.unos3.text.toString()
            val userName = binding.unos4.text.toString()

            database = FirebaseDatabase.getInstance().getReference( path: "Users")

            val User = User(trenTemp,maxTemp,sympBegin,userName)
            database.child(userName).setValue(User).addOnSuccessListener {  it: Void!
                binding.unos1.text.clear()
                binding.unos2.text.clear()
                binding.unos3.text.clear()
                binding.unos4.text.clear()

                Toast.makeText( context: this, text: "Uspješno unešeno",Toast.LENGTH_SHORT).show()
            }.addOnFailureListener {  it: Exception
                Toast.makeText( context: this, text: "Greška",Toast.LENGTH_SHORT).show()
            }


        }

        supportActionBar!!.setDefaultDisplayHomeAsUpEnabled(true)
    }
}
```

Slika 3.24.. *Prikaz koda Podaci.*

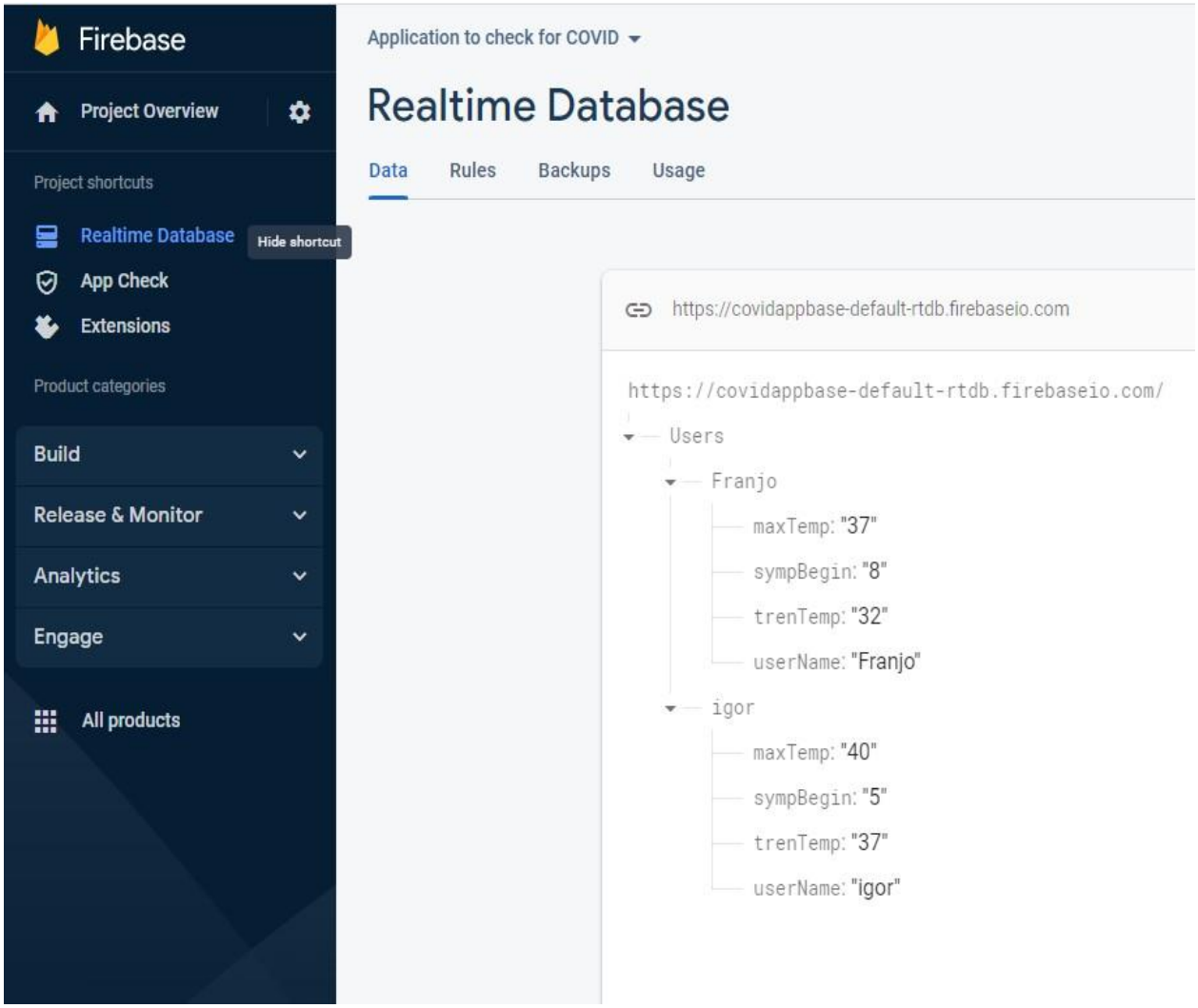
User.kt

```kotlin
package com.example.applicationtocheckforcovid19infectionbasedonsymptoms

import java.text.DecimalFormat

data class User(val trenTemp: String ?= null, val maxTemp: String ?= null, val sympBegin: String ?= null, val userName: String ?=null)
```

Slika 3.25.. *Prikaz koda User.*



Slika 3.26.. *Prikaz spremljenih korisnika u Firebase-u.*

4. ZAKLJUČAK

Cilj završnog rada je bio napraviti android mobilnu aplikaciju koja će na temelju unešenih simptoma korisniku dati povratnu informaciju radi li se o bolesti Covid19 ili nekoj sličnoj bolesti i omogućiti korisniku evidenciju podataka. Aplikacija je napravljena u Android Studio-u koji je besplatno razvojno okruženje dostupno svima. Kod aplikacije je pisan u programskom jeziku Kotlin i svi unešeni podaci od strane korisnika su spremljeni u Firebase. Pojavom novih simptoma korisnik može pomoću aplikacije ponovo provjeriti o kojoj bolesti se radi. Pošto aplikacija ima mogućnost spremanja unešenih podataka, jedan član obitelji može voditi evidenciju za sve ukućane. Aplikacija je jednostavna za korištenje i daje korisniku savjete kako da postupa ovisno o vrsti bolesti. Otvorena je za proširenja ukoliko znanstvenici dođu do novih spoznaja o ovoj zloćudnoj bolesti. Također aplikacija bi se mogla proširiti provjerama za još neke vrste bolesti, drugačijim simptomima i savjetima za liječenje. Uz proširenje klase korisnika User, dodavanjem novih atributa, moguće je i proširiti vrstu podataka koju korisnik sprema u bazu podataka.

LITERATURA

[1] Firebase

https://firebase.google.com/ ,datum zadnjeg pristupa: 3.9.2022.

[2] How to read data from Firebase Realtime Database

https://medium.com/firebase-tips-tricks/how-to-read-data-from-firebase-realtime-database-using-get-269ef3e179c5 ,datum zadnjeg pristupa: 3.9.2022.

[3] CheckBox in Kotlin

https://www.geeksforgeeks.org/checkbox-in-kotlin/ ,datum zadnjeg pristupa: 3.9.2022.

[4] Intents in Kotlin

https://stackoverflow.com/questions/39462397/intents-in-kotlin ,datum zadnjeg pristupa: 3.9.2022.

[5] Develop Android apps with Kotlin

https://developer.android.com/kotlin ,datum zadnjeg pristupa: 3.9.2022.

[6] Kotlin

https://www.w3schools.com/KOTLIN/index.php ,datum zadnjeg pristupa: 3.9.2022.

[7] Plivazdravlje, COVID-19

https://www.plivazdravlje.hr/aktualno/clanak/34384/Gripa-prehlada-ili-COVID-19.html ,datum zadnjeg pristupa: 3.9.2022.

[8] Koronavirus.hr

https://www.koronavirus.hr/sto-moram-znati/o-bolesti/kako-razlikovati-zarazu-koronavirusom-od-prehlade-i-gripe/105 ,datum zadnjeg pristupa: 3.9.2022.

[9] COVID-19: Utvrđivanje simptoma

https://www.health.gov.au/sites/default/files/documents/2020/11/coronavirus-covid-19-utvr-ivanje-simptoma-identifying-the-symptoms.pdf ,datum zadnjeg pristupa: 3.9.2022.

SAŽETAK

Cilj i zadatak završnog rada je bio napraviti android mobilnu aplikaciju koja bi na temelju unosa simptoma od strane korisnika odredila radi li se o bolesti Covid19. Provedeno je istraživanje i formirane su liste s različitim kombinacijama simptoma. Nakon što korisnik unese svoje simptome, simptomi se spremaju u korisničku listu te se ona uspoređuje s listama iz provedenog istraživanja. Nakon što aplikacija usporedi simptome, korisnik dobiva povratnu informaciju o kojoj bolesti se radi i kako treba dalje postupati. Korisnik također ima mogućnost evidencije i pohrane podataka o trenutnoj temperaturi, maksimalnoj dnevnoj temperaturi i broju dana od pojave simptoma. Korisnik može pristupiti svojim podacima unosom predhodno unešenog korisničkog imena. Aplikacija je realizirana u razvojnom okruženju Android Studio, a kod napisan u programskom jeziku Kotlin.

Ključne riječi: Covid19, mobilna aplikacija, simptomi

ABSTRACT

Application to check for COVID19 infection based on symptoms

The goal and task of the assignment were to create an android mobile application that would determine whether it is a disease of Covid19 based on the input of symptoms by the user. The research was conducted and lists were formed with different combinations of symptoms. After the user enters his symptoms, the symptoms are saved in the user list and it is compared with the lists from the conducted research. After the application compares the symptoms, the user receives feedback on which disease it is and how to proceed. The user also has the option of recording and storing data on the current temperature, the maximum daily temperature, and the number of days since the onset of symptoms. The user can access his data by entering the previously entered username. The application was implemented in the Android Studio development environment, and the code was written in the Kotlin programming language.

Key words: Covid19, mobile application, symptoms

PRILOZI