MOBILNA APLIKACIJA ZA PRAĆENJE PARAMETARA I PROCESA PROIZVODNJE VINA

Tokić, Mateo

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

Sveučilišni studij

MOBILNA APLIKACIJA ZA PRAĆENJE PARAMETARA I PROCESA PROIZVODNJE VINA

Završni rad

Mateo Tokić

Osijek, 2022. godina.

SADRŽAJ

1. UVOD 1

1.1 Zadatak završnog rada 1

2. DIGITALNA TEHNOLOGIJA U VINARSTVU I VINOGRADARSTVU 2

2.1 Postojeća web i mobilna rješenja praćenja parametara i procesa proizvodnje vina 2

2.2 Pregled mjernih parametara 3

2.2.1 Vinograd 4

2.2.2 Vinski podrum 4

2.2.3 Vinska bačva 4

2.3 Tijek rada aplikacije 5

3. PROGRAMSKO RJEŠENJE 6

3.1 Radna okolina, programski jezik i tehnologije 6

3.1.1 Razvojno okruženje Android Studio 6

3.1.2 XML opisni jezik 6

3.1.3 Programski jezik Kotlin 6

3.1.4 Firebase 7

3.2 Korisničko sučelje 7

3.3 Programsko rješenje funkcionalnosti aplikacije 12

4. ISPITIVANJE RADA APLIKACIJE 19

4.1 Prijava korisnika 19

4.2 Registracija korisnika 19

4.3 Odabir aktivnosti 20

4.4 Očitavanje parametara 21

4.5 Zapisivanje bilješki 22

5. ZAKLJUČAK 24

LITERATURA 25

SAŽETAK 27

ABSTRACT 28

ŽIVOTOPIS 29

PRILOZI (na USB-u) 30

1. UVOD

Cilj ovog završnog rada je napraviti Android aplikaciju koja je namijenjena korisnicima koji se bave proizvodnjom vina. Ubrzani razvoj mobilnih i računalnih aplikacija uvelike olakšavaju svakodnevne poslove. Svakim danom objavljuju se nove aplikacije koje nude primjenu u gotovo svim okruženjima ljudskog života. Korištenje informacijskih tehnologija u svrhu lakšeg praćenja procesa proizvodnje pružaju korisniku detaljno i smisleno rukovanje mobilnim uređajem u svakom trenutku i na bilo kojemu mjestu, što pridonosi kvaliteti i kvantiteti proizvodnih dobara. Niz faktora potreban je kako bi se osigurala proizvodnja kvalitetnog vina. Stoga, namjena ove aplikacije je praćenje parametara te zapisivanje bilješki koje vinaru pomažu pri procesu proizvodnje i održavanja vina. Točnije rečeno, cilj je ove aplikacije prepoznati mogućnosti nastanka nepovoljnih čimbenika za proizvodnju konstantnim praćenjem procesa u zatvorenim prostorima.

Ovaj rad prikazuje aplikaciju koja korisniku omogućava stvaranje vlastitog profila pomoću kojeg se prijavljuje u sustav. Aplikacija nudi provjeru parametara u vinogradu, vinskom podrumu i bačvi. Osim toga, korisnik može zapisivati vlastite bilješke kako bi što točnije i lakše pratio uvjete sazrijevanja vina.

Drugo poglavlje prikazuje slična postojeća rješenja navedenog problema te opis idejnog rješenja aplikacije. U poglavlju 3 opisano je rješenje, programski jezik i tehnologija korištena za izradu aplikacije. Poglavlje 4 prikazuje rad aplikacije i njezino ispitivanje na primjeru.

1.1 Zadatak završnog rada

U ovom radu potrebno je napraviti Android mobilnu aplikaciju za nadzor i očitavanje parametara u procesu proizvodnje vina. Potrebno je napraviti aplikaciju koja omogućuje prijavu korisniku, praćenje parametara koji se spremaju u *Cloud Firestore* bazu podataka te zapisivanje bilješki vinara.

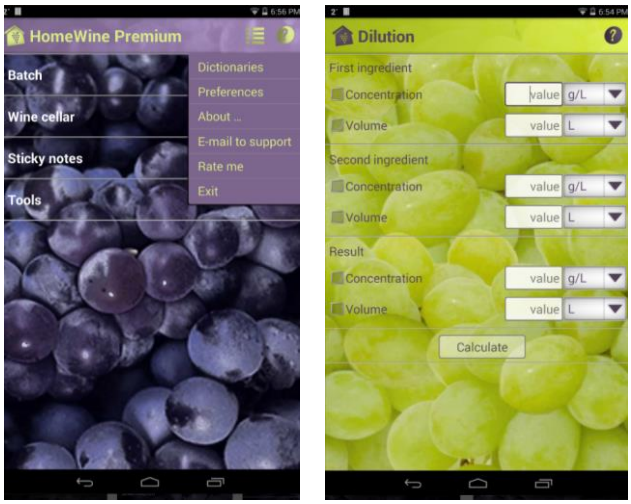
2. DIGITALNA TEHNOLOGIJA U VINARSTVU I VINOGRADARSTVU

Napredak i primjena tehnologije omogućuju vinogradarima da imaju pod kontrolom sve segmente proizvodnje te reagiraju na promjene koje izravno utječu na proizvodnju vina. Uspješnost proizvodnje i uzgoja pojedinih sorti vina u većoj ili manjoj mjeri ovisi o vanjskim čimbenicima koji su povezani s klimatskim i atmosferskim uvjetima i o specifičnostima pojedine kulture. Koristeći digitalnu tehnologiju proizvođači na raspolaganju imaju podatke u stvarnom vremenu te im je tako moguće donošenje odluka koje im pomažu u poboljšanju učinkovitosti i proizvodnji vina veće kvalitete. Primjena takvih tehnologija može značajno unaprijediti proces proizvodnje vina jer se na tržištu pojavljuju razna web i mobilna rješenja koja pomažu u rješavanju problema praćenja i obavljanja poslova vezanih uz vinogradarstvo i vinarstvo.

2.1 Postojeća web i mobilna rješenja praćenja parametara i procesa proizvodnje vina

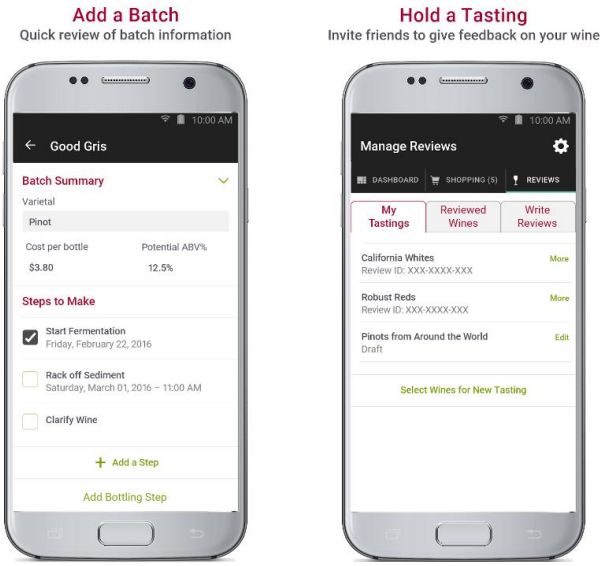
Budući da aplikacija koju ovaj završni rad obrađuje uključuje dvije teme, zapisivanje bilješki i praćenje parametara mikrokontrolera, ona ujedinjuje dvije aplikacije. Na tržištu postoji velik broj aplikacija namijenjenih vođenju bilješki i praćenja parametara, no nekolicina je onih koje omogućuju oboje i to baš u svrhu unaprjeđenja procesa proizvodnje vina. Neke od tih aplikacija su: *HomeWine Premium* i *EnoFile – Home Winemaking*.

HomeWine Premium aplikacija je uglavnom namijenjena za proces proizvodnje vina kod manjih proizvođača vina [1]. Omogućuje korisne funkcije tijekom pripreme serije, procesa fermentacije, ali i za prikupljanje recepata i informacija o proizvedenim vinima (slika 2.1.). *HomeWine Premium* naplaćuje se $2,99 što je možda uzrok malog broj preuzimanja.



Sl. 2.1 Zaslon aplikacije *HomeWine Premium*(slika preuzeta iz [1])

EnoFile – Home Winemaking ima nešto lošije recenzije, ali ima veći broj preuzimanja te se ne naplaćuje. Iz naziva aplikacije vidi se da je također namijenjena proizvodnji kućnih vina. Aplikacija korisniku nudi lakšu kontrolu procesa fermentacije omogućujući mu bilježenje koraka i sastojaka u proizvodnji vina [2]. Uz to uključuje i funkcije za izračun prilagodbe šećera i dodataka SO_2. Kao što je vidljivo na slici 2.2, korisnik može predstaviti svoja vina te pisati recenzije ostalim korisnicima.



Sl. 2.2 Zaslon aplikacije *EnoFile – Home Winemaking*(slika preuzeta iz [2])

2.2 Pregled mjernih parametara

Aplikacija omogućuje korisniku nadziranje parametara vezanih za vinarstvo ali i vinogradarstvo. Stoga, korisniku trebaju rezultati mjerenja parametara iz vinograda, vinskog podruma i bačve u kojoj se vino nalazi.

2.2.1 Vinograd

Prije početka obrade grožđa u vinskom podrumu, bitni su faktori koji utječu na njegovo sazrijevanje u vinogradu. Potrebni su uvjeti u tlu i klimi za normalan rast i razvoj vinove loze i za prinos kvalitetnog grožđa [3]. Klimu vinogorja čine temperatura, sunčeva svjetlost, vlaga, oborine, tlak zraka, vjetar i dr. Vinova loza uvelike ovisi o toplini. Na rast i razvoj vinove loze negativno djeluju temperature niže i više od optimalnih. Niže temperature usporavaju procese rasta, cvatnje i oplodnje, a više usporavaju proces fotosinteze. Za procese stvaranja organske tvari, zagrijavanje tla i zraka, razvoj, rast i redovit prirod vinove loze potrebna je sunčeva svjetlost. Vinovoj lozi je potrebno od 1500 do 2500 sati sunčeve svjetlosti tijekom vegetacije. Za razliku od većine drugih poljoprivrednih biljaka, vinova loza otporna je na sušu, pa se uspješno uzgaja i u krajevima s relativnom malom količinom oborina jer prodirući duboko u tlo snažnim korijenom dolazi do vode i u sušnim područjima. Ipak, nedostatak i loš raspored oborina u svim proizvodnim područjima evidentno smanjuje prinos i kvalitetu grožđa ukoliko vinograd nije navodnjavan. Kad je riječ o vjetrovima, samo blaga strujanja zraka odgovaraju vinovoj lozi. Bržem sušenju suvišne vode i rose s lišća, boljem oprašivanju i oplodnji pridonose lagani vjetrovi koji sprječavaju pojavu kasnih proljetnih mrazova.

2.2.2 Vinski podrum

Vinski podrum skladište je vinskih boca i bačava. Štiti vina, uključujući i šampanjac, od promjene temperature i vlage, previše svjetlosti i drugih vanjskih utjecaja. Tijekom godine podrum mora imati određenu temperaturu i vlažnost zraka, posebno u dijelu gdje vino dozrijeva u bačvama i bocama [4]. U podzemnom je podrumu temperatura tijekom godine približno stalna, oko 11.5°C. Hlapljenje vina, pojava gljivica i plijesni, ali i korozija svih kovinskih predmeta u podrumu ovisi o relativnoj vlažnosti zraka. Optimalna vlažnost je između 65 i 70% pri temperaturi od 12 do 15°C. Razvoj plijesni na zidovima uzrokuje i udio CO_2 i količina svjetlosti u podrumu. Vino je potrebno držati podalje od jakih, izravnih izvora svjetlosti ako vino treba čuvati dulje od godinu dana [5]. Izravna svjetlost može pokvariti vino reagirajući s njegovim fenolnim spojevima, posebno u vinima lakog tijeka.

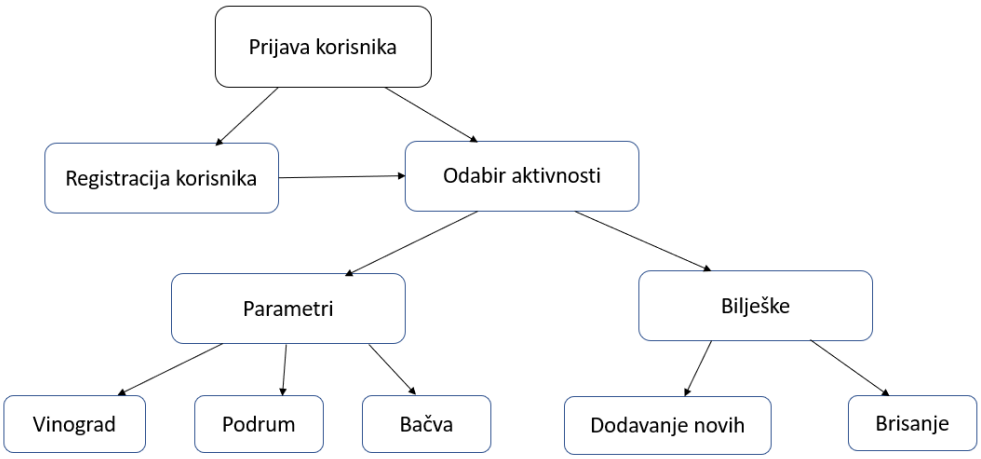
2.2.3 Vinska bačva

U vinskim bačvama vino prolazi kroz procese dozrijevanja te je potrebno vino pravovremeno pretočiti s taloga kako ne bi poprimilo loš miris i ukus. Vino je potrebno pretočiti u studenom ili do prve polovice prosinca, kako bi ostalo dovoljno vremena da se izbistri i filtrira [6]. Najbitniji parametri u procesu dozrijevanja vina u vinskoj bačvi su: šećer, kiselina, sumpor, prozirnost, gustoća, vlaga, boja, CO_2, temperatura i tlak. Šećer je jedan od najvažnijih sastojaka

grožđa [7]. Tijekom alkoholne fermentacije (vrenja) djelovanjem vinskih kvasaca dolazi do pretvorbe šećera u etilni alkohol. Ovisno o sorti i kakvoći udio alkohola u vinu iznosi od 8,5 do 15vol.%. Prema Pravilniku o vinu maksimalno je dopušteno dodati do 2,5g/l vinske kiseline i do 1 g/l limunske kiseline, ali ukupna se kiselost ne smije povisiti za više od 2,5 g/l.

2.3 Tijek rada aplikacije

Ovakva mobilna aplikacija osigurava korisniku prijavu u sustav te samostalno i efektivno praćenje uvjeta u vinogradu, vinskom podrumu i bačvi. Time se omogućava pravovremeno i valjano djelovanje ukoliko prethodno navedeni parametri iznosom mogu naštetiti kvalitetnom dozrijevanju grožđa i vina. Aplikacija nudi prijavu korisnika u sustav pomoću adrese e-pošte i lozinke. Ukoliko korisnik nije prijavljen, omogućena mu je registracija stvaranjem novog profila pomoću kojeg će se ubuduće prijavljivati u sustav. Zatim, korisnik bira želi li očitati parametre spremljene u bazi podataka ili zapisati svoje bilješke. Odabirom na očitanje parametara na zaslonu aplikacije korisnik vidi listu parametara koji se odnose na vinograd, a klizanjem po zaslonu prema desno na zaslonu se mijenjaju parametri u vinskom podrumu te parametri u vinskoj bačvi. Osim klizanjem po zaslonu, na zaslonu se nalaze kartice s natpisima *Vineyard*(eng. Vinograd), *Wine Cellar*(eng. Vinski podrum) i *Wine Barrel*(eng. Vinska bačva). Korisnikovim odabirom aktivnosti zapisivanja bilješki, na zaslonu se otvara popis bilješki koje su spremljene u bazi podataka te ih korisnik može brisati ili dodavati nove. U gornjem desnom uglu zaslona nalazi se izbornik koji korisniku nudi opcije odjave i povratka na zaslon za odabir aktivnosti pisanja bilješki ili očitanja parametara. Na slici 2.3 prikazan je dijagram koji prikazuje korištenje mobilne aplikacije.



Sl. 2.3 Dijagram korištenja aplikacije

3. PROGRAMSKO RJEŠENJE

U ovom poglavlju završnog rada bit će opisano potrebna radna okolina i potrebno znanje za razvoj aplikacije. Bit će objašnjeno *Android Studio* razvojno okruženje, dizajniranje korisničkog sučelja XML opisnim jezikom, funkcionalnosti aplikacije napisane Kotlin programskim jezikom te simulacija aplikacije Android virtualnim uređajem.

3.1 Radna okolina, programski jezik i tehnologije

Za razvoj ove aplikacije potrebno je poznavanje razvojnog okruženja Android Studio, programskog jezika Kotlin i određeno znanje iz vinogradarstva i vinarstva.

3.1.1 Razvojno okruženje Android Studio

Android Studio službeno je integrirano razvojno okruženje (IDE) za Googleov operativni sustav Android, dizajnirano posebno za razvoj Android aplikacija [8] i izgrađeno na *JetBrainsovom* IntelliJ IDEA softveru. Preuzimanje je dostupno na Windows, macOS i Linux operativnim sustavima. Android Studio najavljen je na Google I/O konferenciji 16. svibnja 2013., a prva stabilna verzija izdana je u prosincu 2014. godine. 7. svibnja 2019. godine Kotlin je zamijenio Javu kao Googleov preferirani jezik za razvoj Android aplikacija. Unatoč tome, Android Studio podržava i programske jezike Javu i C++. Da bi se Android projekt pokrenuo, potrebno je spojiti i uključiti uređaj. Osim pokretanja na stvarnom uređaju, Android studio omogućuje pokretanje projekta na virtualnom uređaju. *Android Virtual Device* (AVD).

3.1.2 XML opisni jezik

XML (*EXtensible Markup Language*) opisni je jezik i format datoteke za pohranu, prijenos i rekonstrukciju proizvoljni podataka. Definira skup pravila za kodiranje dokumenata u format čitljiv i ljudima i računalnim programima. Format oznaka XML jezika sličan je formatu oznaka HTML jezika. Korištenje je vrlo jednostavno: odgovarajućim oznakama, koje imaju poznato i lako shvatljivo značenje, potrebno je uokviriti odgovarajući sadržaj koji one opisuju.

3.1.3 Programski jezik Kotlin

Kotlin je statički objektno orijentirani programski jezik opće namjene prvobitno dizajniran za JVM (*Java Virtual Machine*) i Android [9]. Nastao je 2010. godine u *JetBrainsu*, a od 2012. postao je programom otvorenog koda. Budući da se pokreće pomoću Java virtualne mašine, može se kompilirati u Java byte kod. Osim *Jave*, moguće je kompiliranje i u *JavaScript* kod te se zbog toga može koristiti i za razvoj više-platformskih aplikacija. Kotlin pruža mnoge prednosti kako bi pojednostavio i ubrzao pisanje koda.

3.1.4 Firebase

Firebase je platforma koju je razvio *Google* za izradu web i mobilnih aplikacija. *Firebase* je osnovan kao neovisna tvrtka 2011. godine, a 2014. godine *Google* je kupio platformu. Firebaseov prvi proizvod bila je *Firebase Realtime Database*, API koji sinkronizira podatke u stvarnom vremenu aplikacija na iOS, Android i web uređajima i pohranjuje ih u Firebaseov oblak. Sljedećih godina *Firebase* je objavio velik broj drugih proizvoda kao što su: . *Firebase Hosting*, *Firebase Authentication*, *Firebase Cloud Messaging*, *Firestore Database* i mnoge druge [10]. *Firebase* je u listopadu 2017. godine pokrenuo *Cloud Firestore*, bazu podataka dokumenata u stvarnom vremenu koja nasljeđuje *Firebase Realtime Database*.

3.2 Korisničko sučelje

Resursima koji se nazivaju *layout* definiran je izgled Android aplikacije. U mapi layout nalaze se datoteke napisane XML opisnim jezikom kojim se definiraju elementi korisničkog sučelja, tj. izgled zaslona aplikacije. Glavna klasa svake Android aplikacije zove se *Activity*, a predstavlja jedan zaslon aplikacije. Unutar aktivnosti moguće je postaviti *Fragmente*, Android komponente koje sadrže određeno ponašanje ili dio korisničkog sučelja.

Aplikacija sadrži pet *Activitya*, početni *LogInActivity* koji služi za prijavu korisnika, *SignUpActivity* za registraciju korisnika, *SelectActivity* za odabir očitavanja parametara ili zapisivanja bilješki, *MainActivity* u kojem su uključena 3 fragmenta (*VineyardFragment*, *WineCellarFragment*, *WineBarrelFragment*) koji služe za praćenje parametara, te *NotesActivity* za pisanje bilješki.

```
<EditText
    android:id="@+id/edt_email"
    android:layout_width="match_parent"
    android:layout_height="50dp"
    android:layout_below="@id/app_logo"
    android:layout_marginTop="10dp"
    android:hint="Email"
    android:paddingLeft="13dp"
    android:background="@drawable/edt_background"
    android:layout_marginLeft="20dp"
    android:layout_marginRight="20dp"
    android:autofillHints="" />

<EditText
    android:id="@+id/edt_password"
    android:layout_width="match_parent"
    android:layout_height="50dp"
    android:layout_below="@id/edt_email"
    android:layout_marginTop="10dp"
    android:hint="Password"
    android:paddingLeft="13dp"
    android:background="@drawable/edt_background"
    android:layout_marginLeft="20dp"
    android:layout_marginRight="20dp"
    android:autofillHints="" />
```

Sl. 3.1 Polja za unos email adrese i lozinke korisnika

Opisni kod na slici 3.1 prikazuje dvije *EditText* klase koje predstavljaju UI kontrolu za unos niza znakova. Prvi *EditText* služi za unos korisnikove email adrese, a drugi za unos lozinke. Svaki *EditText* mora sadržavati svojstvo *id* koje se mora razlikovati od ostalih *id* svojstava drugih *layout* klasa kako bi pri pisanju programskog koda mogli razlikovati pojedine elemente sučelja. Svojstvima *layout_width* i *layout_height* definirana je širinu i visinu ovog elementa. Vrijednost *match_parent* označuje da se širina prilagođava širini roditeljske klase, u ovom slučaju *RelativeLayout* klase koja zauzima širinu cijeloga zaslona. Mjerna jedinica *dp* (*Density-independent Pixels*) apstraktna je jedinica koja se temelji na fizičkoj gustoći zaslona. Unutar mape drawable pohranjuju se slike i ostali koncepti koji se mogu opisati XML kodom ili nacrtati na zaslonu, a moguće ih je primijeniti na drugim XML resurs atributima. Tako je u obje EditText klase atributu *android:drawable* dodana vrijednost *@drawable/edt_background* koja definira izgled ovih polja. Kod koji opisuje *edt_background* koncept nalazi se na slici 3.2.

```xml
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android">
    <corners android:radius="20dp"/>
    <stroke android:color="@color/button" android:width="3dp"/>
</shape>
```

Sl. 3.2 Koncept *edt_backgorund*

Osim *EditText* klase, *LogInActivity* sadrži i *ImageView* klasu koja služi za prikaz loga aplikacije, *Button* klasu s *text* atributom vrijednosti „Log in“ koja omogućuje prijavu korisnika te *Button* klasu s *text* atributom „Sign up“ koja omogućuje otvaranje aktivnosti za registraciju korisnika.

```xml
<Button
    android:id="@+id/btnSignUp"
    android:layout_width="150dp"
    android:layout_height="wrap_content"
    android:layout_below="@id/edt_password"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="10dp"
    android:background="@drawable/btn_background"
    android:text="Sign up"
    android:textColor="@color/white"
    android:textSize="20sp" />

</RelativeLayout>
```

Sl. 3.3 Gumb za registraciju

Izgled zaslona *SignUpActivity* sličan je početnom *LogInActivity* zaslonu te sadrži tri *EditText* klase, za unos imena, email adrese i lozinke korisnika, te jednu *Button* klasu kojom se korisnik registrira. *Button* klasa predstavlja gumb čiji je kod prikazan na slici 3.3.

Izgled *SelectActivitya* izrađen je pomoću *GridLayout* koji sve ugniježđene *View* elemente postavlja u pravokutnu mrežu. Mreža se sastoji od skupa tankih linija koje dijele zaslon na ćelije. Tako je *GridLayout* podijeljen na dva stupca pridruživanjem vrijednosti „2“ atributu *android:columnCount*, i 1 red pridruživanjem vrijednosti 1 atributu *android:rowCount*. Prvoj ćeliji pristupa se atributima *android:layout_rowWeight android:layout_columnWeight*. Prva ćelija je *CardView* element unutar kojeg se nalazi *LinearLayout* u kojem su smješteni *ImageView* klasu za prikaz slike te *TextView* s natpisom „STATUS“ koji simbolizira očitavanje parametara te je namijenjen odabiru *MainActivitya*. Kod koji opisuje prvi *CardView* prikazan je na slici 3.4.

```
<androidx.cardview.widget.CardView
    android:backgroundTint="#45B39D"
    android:id="@+id/status"
    android:layout_height="0dp"
    android:layout_width="0dp"
    app:cardCornerRadius="15dp"
    android:layout_rowWeight="1"
    android:layout_columnWeight="1"
    app:cardElevation="0dp"
    android:layout_margin="5dp">
    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="vertical"
        android:gravity="center"
        android:layout_gravity="center_vertical|center_horizontal">
        <ImageView
            android:layout_width="wrap_content"
            android:layout_height="100dp"
            android:src="@drawable/setting"/>
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="STATUS"
            android:layout_marginTop="20dp"
            android:textSize="20sp"
            android:textAlignment="center"
            android:textStyle="bold"/>
    </LinearLayout>
</androidx.cardview.widget.CardView>
```

Slika 3.4 *CardView* za odabir *MainActivitya*

Drugi *CardView* element u kojem se nalazi *TextView* s natpisom „NOTES“ namijenjen je odabiru *NotesActivitya*. *CardView* elementi zamjenjuju *Button* klase za odabir sljedećih aktivnosti.

Slika 3.5 prikazuje kod koji opisuje izgled zaslona *MainActivitya*. *MainActivitya*sadrži tri fragmenta s popisom parametara i *TabLayout* koji služi za prikaz kartica s imenom fragmenta. Da bi se omogućila izmjena fragmenata listanjem preko zaslona, *MainActivity* sadržava *ViewPager* klasu koja pruža funkcionalnost okretanja stranica u aplikaciji.

```
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity"
    android:background="@color/background">

    <com.google.android.material.tabs.TabLayout
        android:id="@+id/tab"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintBottom_toTopOf="@id/view_pager"
        android:background="@color/tab"/>

    <androidx.viewpager.widget.ViewPager
        android:id="@+id/view_pager"
        android:layout_width="0dp"
        android:layout_height="0dp"
        app:layout_constraintTop_toBottomOf="@id/tab"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

Sl. 3.5 Sučelje *MainActivitya*

Sučelje svih fragmenata slično je opisano. Sastoji se od onoliko *LinearLayouta* koliko je parametara potrebno očitati u vinogradu, vinskom podrumu i vinskoj bačvi. U *VineyardFragmentu* prikazuje se šest parametara očitanih u vinogradu. U *WineCellarlFragmentu* nalaze se četiri *LinearLayouta* jer je potrebno prikazati četiri parametra očitanih iz vinskog podruma. Budući da je deset parametara potrebno očitati iz vinske bačve, *WineBarrelFragment* sadrži deset *LinearLayouta*. U svaki *LinearLayout* ugniježđene su dvije *TextView* klase. Prvi *TextView* prikazuje naziv parametra, a drugi prikazuje vrijednost spremljenu u *Cloud Firestore* bazu podataka. Opisni kod *LinearLayouta* koji se nalazi u *VineyardFragmentu* a prikazuje temperaturu u vinogradu prikazan je na slici 3.6.

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_gravity="center_vertical"
    android:layout_marginTop="0dp"
    android:background="@color/vineyard_color"
    android:orientation="horizontal">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="40dp"
        android:layout_marginStart="10dp"
        android:layout_marginTop="10dp"
        android:text="Temperature:"
        android:textColor="@color/text"
        android:textSize="22sp"
        android:textStyle="bold" />
    <TextView
        android:id="@+id/temperature1"
        android:layout_width="wrap_content"
        android:layout_height="40dp"
        android:layout_marginStart="10dp"
        android:layout_marginTop="10dp"
        android:textColor="@color/text"
        android:textSize="20sp" />
    <ImageView
        android:layout_width="50dp"
        android:layout_height="50dp"
        android:layout_marginStart="10dp"
        android:background="@drawable/sun" />
</LinearLayout>
```

Sl. 3.6 Prikaz jednog parametra

Slika 3.7 prikazuje kod sučelja *NotesActivityja*. Na dnu zaslona se nalaze dva gumba te jedna *EditText* klasa koja služi za upisivanje bilješki korisnika. Gumb koji sadrži tekst „Delete“ služi za brisanje bilješki, a gumb „Add“ za dodavanje bilješki napisanih u *EditText* klasi.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".NotesActivity"
    android:background="@color/background">

    <androidx.recyclerview.widget.RecyclerView
        android:id="@+id/rvNotes"
        android:layout_width="match_parent"
        android:layout_height="match_parent">

    </androidx.recyclerview.widget.RecyclerView>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignBottom="@+id/rvNotes"
        android:layout_marginBottom="10dp">

        <Button
            android:id="@+id/deleteNote"
            android:layout_width="75dp"
            android:layout_height="wrap_content"
            android:text="Delete"
            android:textColor="@color/white"
            android:background="@drawable/btn_background"
            android:layout_marginLeft="3dp"/>


    <EditText
        android:id="@+id/inputNote"
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:ems="11"
        android:layout_marginLeft="3dp"
        android:hint="Parameter: Value"
        android:paddingLeft="13dp"
        android:background="@drawable/edt_background"
        android:textColor="@color/text"
        tools:layout_editor_absoluteX="100dp"
        tools:layout_editor_absoluteY="673dp" />

    <Button
        android:id="@+id/addNote"
        android:layout_width="75dp"
        android:layout_height="wrap_content"
        android:layout_marginRight="3dp"
        android:layout_marginLeft="3dp"
        android:background="@drawable/btn_background"
        android:text="Add"
        android:textColor="@color/white"
        />

    </LinearLayout>
</RelativeLayout>
```

Sl. 3.7 Sučelje *NotesActivityja*

RecyclerView je UI komponenta koja olakšava učinkovito prikazivanje velikih skupova podataka, tj. listu sadržaja [11]. Elementi koji izađu iz prikaza recikliraju se s novim sadržajem i prikazuju korisniku na zaslonu. Za korištenje *RecylcerViewa* potrebno je definirati izgled stavke, u ovom slučaju bilješki, a biblioteka *RecyclerView* dinamički stvara elemente kada su potrebni. Izgled bilješki definiran je dodavanjem nove *Layout Resource* datoteke pod nazivom *note_item.xml* čiji kod je prikazan na slici 3.8.

```xml
<?xml version="1.0" encoding="utf-8"?>
<androidx.cardview.widget.CardView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:backgroundTint="@color/background"
    xmlns:app="http://schemas.android.com/apk/res-auto">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_margin="5dp"
        android:background="@drawable/btn_background"
        android:backgroundTint="#A9DFBF">
        <TextView
            android:id="@+id/noteTitle"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="10dp"
            android:layout_marginLeft="10dp"
            android:layout_marginBottom="10dp"
            android:text="Parameter: Value"
            android:textColor="@color/text"
            android:textSize="25sp"
            />
    </LinearLayout>
</androidx.cardview.widget.CardView>
```

Sl. 3.8 *note_item.xml*

3.3 Programsko rješenje funkcionalnosti aplikacije

Android Studio odvaja funkcionalnosti od prikaza stoga je izgled (*layout*) definiran unutar layout (*.xml*) datoteka, a sva funkcionalnost aplikacije nalazi se unutar kotlin (*.kt*) datoteka. Funkcionalnosti svakog *activityja* i fragmenta pišu se *Kotlin* programskim kodom u datoteke koje nose naziv kao i *activity* s nastavakom *.kt*.

```kotlin
class LogInActivity : AppCompatActivity() {

    private lateinit var edtEmail: EditText
    private lateinit var edtPassword: EditText
    private lateinit var btnLogin: Button
    private lateinit var btnSignUp: Button
    private lateinit var mAuth: FirebaseAuth

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_log_in)

        supportActionBar?.hide()

        mAuth = FirebaseAuth.getInstance()

        edtEmail = findViewById(R.id.edt_email)
        edtPassword = findViewById(R.id.edt_password)
        btnLogin = findViewById(R.id.btnLogIn)
        btnSignUp = findViewById(R.id.btnSignUp)

        btnSignUp.setOnClickListener{ it: View!
            val intent = Intent( packageContext: this, SignUpActivity::class.java)
            startActivity(intent)
        }
        btnLogin.setOnClickListener{ it: View!
            val email = edtEmail.text.toString()
            val password = edtPassword.text.toString()
            login(email,password)
        }
    }
```

Sl. 3.9. *LogInActivity.kt*

Kako je prikazano na slici 3.9., unutar *onCreate()* metode inicijalizirani su svi gumbi i *EditText* klase jer se ta metoda izvršava prilikom pokretanja aplikacije. Unutar metoda

setOnClickListener(), koje služe za reagiranje na pritisak gumba, inicijalizirani su *Intent* objekti pomoću kojih je omogućen prelazak na drugi *activity*. Osim toga, dodana je funkcija *login* koja za predane joj vrijednosti email adrese i lozinke izvršava prijavu korisnika. Ugrađena metoda *signInWithEmailAndPassword()* uspoređuje predane joj argumente (email adresa i lozinka) s onima u *Firebase Authentication* bazi podataka. Programski kod funkcije *login* prikazan je na slici 3.10. *Toast* porukom korisnik se obavještava da nije registriran.

```
private fun login(email: String, password: String){
    mAuth.signInWithEmailAndPassword(email, password)
        .addOnCompleteListener(this) { task ->
            if (task.isSuccessful) {
                val intent = Intent( packageContext: this@LogInActivity, SelectActivity::class.java)
                finish()
                startActivity(intent)

            } else {
                Toast.makeText( context: this@LogInActivity,  text: "User does not exist", Toast.LENGTH_SHORT).show()
            }
        }
}
```

Sl. 3.10 Funkcija *login()*

Unutar *onCreate()* metode *SignUpActivityja* također je potrebno inicijalizirati sve gumbe i *EditText* klase, a unutar *setOnClickListener()* metode gumba koji služi za registraciju korisnika poziva se funkcija *signup* čiji je programski kod prikazan na slici 3.11. Ugrađena metoda *createUserWithEmailAndPassword()* registrira korisnika s upisanom email adresom i lozinkom te *startActivity()* funkcijom pokreće *SelectActivity*.

```
private fun signup(email: String, password: String){
    mAuth.createUserWithEmailAndPassword(email, password)
        .addOnCompleteListener(this) { task ->
            if (task.isSuccessful) {
                val intent = Intent( packageContext: this@SignUpActivity, SelectActivity::class.java)
                finish()
                startActivity(intent)
            } else {
                Toast.makeText( context: this@SignUpActivity,  text: "Some Error", Toast.LENGTH_SHORT).show()
            }
        }
}
```

Sl. 3.11 Funkcija *signup()*

SelectActivity sadrži dva *Intent* objekta koji omogućuju prijelaz na sljedeće *activityje*. Pritiskom na lijevi dio zaslona s natpisom „STATUS“ otvara se *MainActivity*, a pritiskom na desni dio zaslona s natpisom „NOTES“ otvara se *NotesActivity* s mogućnosti zapisivanja bilješki.

Slika 3.12 prikazuje programsko rješenje ovog problema metodama *setOnClickListener()* u kojima su inicijalizirani *Intent* objekti.

```
status.setOnClickListener{ it: View!
    var intent = Intent( packageContext: this, MainActivity::class.java)
    startActivity(intent)
}
notes.setOnClickListener{ it: View!
    var intent = Intent( packageContext: this, NotesActivity::class.java)
    startActivity(intent)
}
```

Sl. 3.12 Metode za prijelaz na drugi *activity*

Korisnik ima mogućnost odjave i povratka na *SelectActivity* pritiskom na izbornik. Na slici 3.13. prikazan je kod implementacije izbornika u klasu *activityja*.

```
override fun onCreateOptionsMenu(menu: Menu): Boolean {
    menuInflater.inflate(R.menu.menu, menu)
    return super.onCreateOptionsMenu(menu)
}

override fun onOptionsItemSelected(item: MenuItem): Boolean {
    if(item.itemId == R.id.logout){
        mAuth.signOut()
        val intent = Intent( packageContext: this@MainActivity, LogInActivity::class.java)
        finish()
        startActivity(intent)
        return true
    }
    if(item.itemId == R.id.select){
        val intent2 = Intent( packageContext: this@MainActivity, SelectActivity::class.java)
        startActivity(intent2)
        return true
    }
    return true
}
```

Sl. 3.13 Metode implementacije izbornika

Prvom metodom *onCreateOptionsMenu()* [12] inicijaliziran je sadržaj izbornika tako što mu je dodan resurs smješten u mapi *menu* koji je definiran XML opisnim jezikom (slika 3.14). Drugu metodu *onOptionsItemSelected()* sustav poziva kada korisnik odabere stavku iz izbornika opcija. Ova metoda prosljeđuje odabranu stavku izborniku.

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/select" android:title="Select"/>
    <item android:id="@+id/logout" android:title="Log out"/>
</menu>
```

Sl. 3.14 *menu.xml*

MainActivity sadrži *ViewPager* objekt koji ima ugrađene geste prelaska prstom za prijelaz kroz stranice te koristi *PageAdapter* objekt kao izvor nove stranice za prikaz. Na slici 3.15 prikazan je programski kod koji opisuje klasu *PageAdapter* koja nasljeđuje apstraktnu klasu

FragmentPagerAdapter. *PageAdapter* klasa implementira dvije metode: *getItem()* koja vraća instance fragmenata i metodu *getCount()* koja vraća broj stranica koje adapter kreira.

```
class PageAdapter(var context: Context, fm: FragmentManager, var totalTabs: Int):
    FragmentPagerAdapter(fm) {

    override fun getItem(position: Int): Fragment {
        return when(position){
            0 -> { VineyardFragment() }
            1 -> { WineCellarFragment() }
            2 -> { WineBarrelFragment() }
            else ->  getItem(position)
        }
    }

    override fun getCount(): Int {
        return totalTabs
    }
}
```

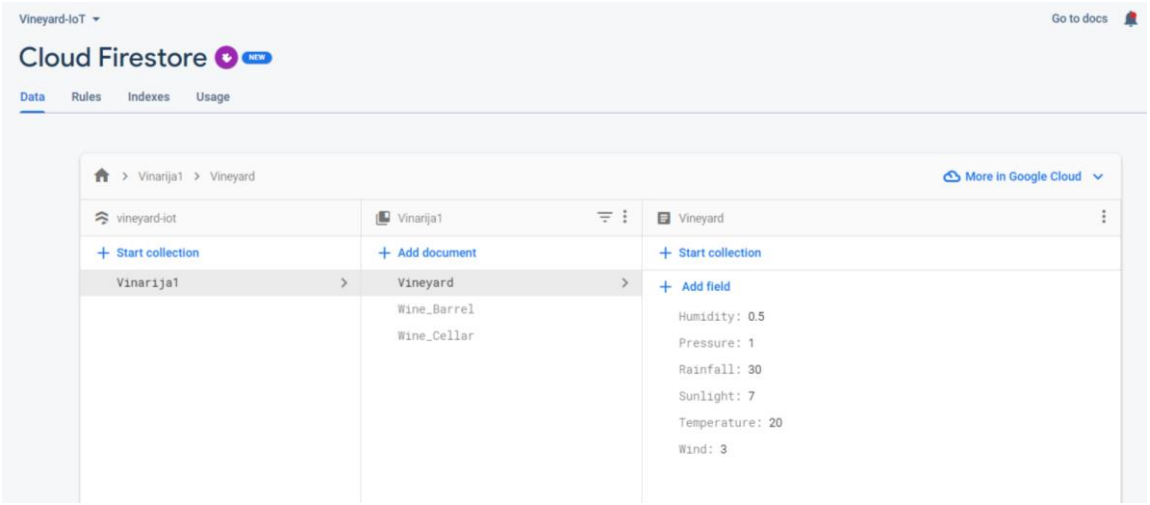
Sl. 3.15 *PageAdapter* klasa

U *MainActivityju* potrebno je dodati tri nove kartice. Kartice nose nazive „Vineyard“ (Vinograd), „Wine cellar“ (Vinski podrum) i „Wine barrel“ (Vinska bačva) kako bi se naznačilo mjesto očitanih parametara. Slika 3.16 prikazuje dodavanje kartica te postavljanje adaptera za *ViewPager*. Metoda *addOnPageChangeListener()* poziva se kad god se promijeni adapter za *ViewPager*, tj. svaki puta kada se promijeni stranica. Metoda *onTabSelectedListener()* poziva se kada se promijeni odabir kartice te se tako omogućuje promjena stranica.

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)

    mAuth = FirebaseAuth.getInstance()

    viewPager = findViewById(R.id.view_pager)

    tabLayout = findViewById(R.id.tab)
    tabLayout.addTab(tabLayout.newTab().setText("Vineyard"))
    tabLayout.addTab(tabLayout.newTab().setText("Wine cellar"))
    tabLayout.addTab(tabLayout.newTab().setText("Wine barrel"))
    tabLayout.tabGravity=TabLayout.GRAVITY_FILL

    val adapter = PageAdapter( context: this, supportFragmentManager, tabLayout.tabCount)

    viewPager.adapter = adapter
    viewPager.addOnPageChangeListener(TabLayout.TabLayoutOnPageChangeListener(tabLayout))
    tabLayout.addOnTabSelectedListener(object: TabLayout.OnTabSelectedListener{
        override fun onTabSelected(tab: TabLayout.Tab?) {
            viewPager.currentItem= tab!!.position
        }
        override fun onTabUnselected(tab: TabLayout.Tab?) {
        }
        override fun onTabReselected(tab: TabLayout.Tab?) {
        }
    })
}
```

Sl. 3.16 *onCreate()* metoda *MainActivity.kt datoteke*

Svaki fragment prikazuje parametre spremljene u *Cloud Firestore* bazi podataka. Da bi se omogućilo dohvaćanje podataka potrebno je napraviti novu kolekciju u kojoj se nalaze tri dokumenta s nazivima „Vineyard“, „Wine_Cellar“ i „Wine_Barrel“. U svaki dokument dodano je polje s nazivom parametara potrebnih za očitanje. Slika 3.17 prikazuje bazu podataka nazvanu „Vinarija1“ te parametre spremljene u dokument „Vineyard“ koji označuju mjerenja u vinogradu.



Sl. 3.17 Baza podataka s parametrima

Na slici 3.18 prikazan je programski kod *VineyardFragment.kt* datoteke kojim se dohvaćaju parametri iz baze podataka. Preko instance *FirebaseFirestore* objekta dohvaća se kolekcija metodom *collection()* koja kao argument prima naziv baze podataka. Metodom *document()* kojoj se kao argument predaje naziv dokumenta dohvaćamo dokument koji označuje naziv mjesta očitanja parametra. Metodom *get()* dohvaćaju se vrijednosti polja.

```
override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    super.onViewCreated(view, savedInstanceState)
    db = FirebaseFirestore.getInstance()

    settextview(view)
    val docRef = db.collection( collectionPath: "Vinarija1").document( documentPath: "Vineyard")
    docRef.get()
        .addOnSuccessListener { document ->
            if (document != null) {
                temperature.text = document.get("Temperature").toString() + "°C"
                rainfall.text = document.get("Rainfall").toString() + "mm/m^2"
                humidity.text = document.get("Humidity").toString() + "kg/m^3"
                pressure.text = document.get("Pressure").toString() + "bar"
                sunlight.text = document.get("Sunlight").toString() + "h"
                wind.text = document.get("Wind").toString() + "km/h"
            } else {
                Log.d(ContentValues.TAG, msg: "No such document")
            }
        }
        .addOnFailureListener { exception ->
            Log.d(ContentValues.TAG, msg: "get failed with ", exception)
        }
}
```

Sl. 3.18 *VineyardFragment.kt*

Ukoliko korisnik na *SelectActivityju* odabere aktivnost zapisivanja bilješki, na zaslonu se prikazuje sučelje *NotesActivityja*. Da bismo mogli zapisane bilješke prikazati na zaslonu, potrebno je stvoriti novu Kotlin Data Class datoteku koja treba predstavljati klasu bilješke (slika 3.19).

```
data class Note(
    val Title: String
)
```

Sl. 3.19 *Note.kt*

Budući da se bilješke prikazuju na zaslonu pomoću *RecyclerViewa*, potrebno je napisati adapter klasu koja omogućuje prilagodbu prikaza sadržaja iz objekata modela u *View* objekte. Osim adapter klase potrebno je napisati i *ViewHolder* klasu koja predstavlja jedan element unutar *RecyclerViewa*. *ViewHolder* klasu moguće je definirati unutar adapter klase kao što je prikazano na slici 3.20. *NotesAdapter* klasa sadrži tri metode koje se moraju upotrijebiti kako bi funkcionirala. *getItemCount()* metoda vraća ukupni broj elemenata koji se nalaze u *RecyclerViewu*. Metoda *onCreateViewHolder()* služi za kreiranje i recikliranje *ViewHoldera*, a metoda *onBindViewHolder()* služi za povezivanje *ViewHoldera* i podataka koji se nalaze u adapteru.

```
class NotesAdapter(private val noteList:ArrayList<Note>) : RecyclerView.Adapter<NotesAdapter.ViewHolder>() {
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): NotesAdapter.ViewHolder {
        val view = LayoutInflater.from(parent.context).inflate(
            R.layout.note_item, parent,  attachToRoot: false)
        return ViewHolder(view)
    }
    override fun onBindViewHolder(holder: NotesAdapter.ViewHolder, position: Int) {
        var newNote = noteList[position]
        holder.bind(newNote)
    }
    override fun getItemCount(): Int {
        return noteList.size
    }
    class ViewHolder constructor(itemview: View) : RecyclerView.ViewHolder(itemview) {
        private val noteTitle = itemview.findViewById<TextView>(R.id.noteTitle)
        fun bind(note: Note){
            noteTitle.text = note.Title
        }
    }
    fun addNote(title: String, position: Int) {
        var noteInfo = Note(title)
        if(noteList.size >= position) {
            noteList.add(position, noteInfo)
            notifyItemInserted(position)
        }
    }
    fun clearAll(){
        noteList.clear()
        notifyDataSetChanged()
    }
}
```

Sl. 3.20 *NotesAdapter* klasa

Funkcije *addNote()* i *clearAll()* služe za dodavanje, odnosno brisanje bilješki iz *RecyclerViewa*. Unutar *NotesActivityja* u *onCreate()* metodi potrebno je inicijalizirati sve gumbe, *EditText* klasu, *RecyclerView* i *NotesAdapter* objekte. Na gumbe *addNote* i *deleteNote* dodane su *setOnClickListener()* metode kojima se omogućuje dodavanje i brisanje bilješki dodirom gumba. Slika 3.21 prikazuje programski kod *onCreate()* metode datoteke *NotesActivity*.

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_notes)

    mAuth = FirebaseAuth.getInstance()

    rv = findViewById(R.id.rvNotes)
    rv.layoutManager = LinearLayoutManager( context: this)
    rv.setHasFixedSize(true)

    noteArrayList = arrayListOf()
    noteAdapter = NotesAdapter(noteArrayList)
    rv.adapter = noteAdapter

    deleteNote = findViewById(R.id.deleteNote)
    addNote = findViewById(R.id.addNote)
    inputNote = findViewById(R.id.inputNote)

    addNote?.setOnClickListener { it: View!
        val noteTitle = inputNote.text.toString()
        noteAdapter.addNote(noteTitle, noteArrayList.size)
    }
    deleteNote?.setOnClickListener { it: View!
        noteAdapter.clearAll()
    }
}
```

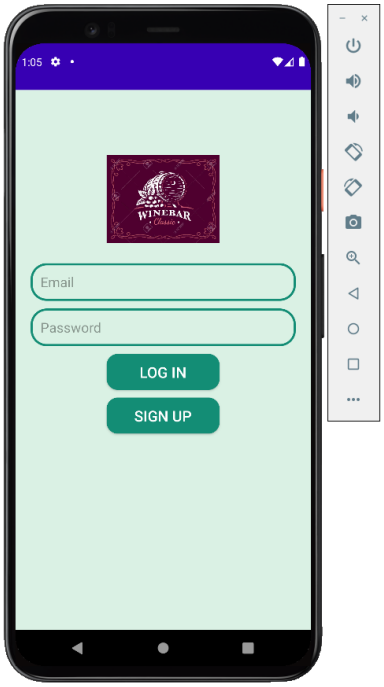
Sl. 3.21 *onCreate()* metoda *NotesActivity.kt datoteke*

4. ISPITIVANJE RADA APLIKACIJE

U ovom poglavlju je opisan postupak ispitivanja rada aplikacije njezinim korištenjem te provjera rada baze podataka. Ispitivanje je izvedeno pokretanjem aplikacije na virtualnom uređaju, *Android Virtual Device* (AVD).

4.1 Prijava korisnika

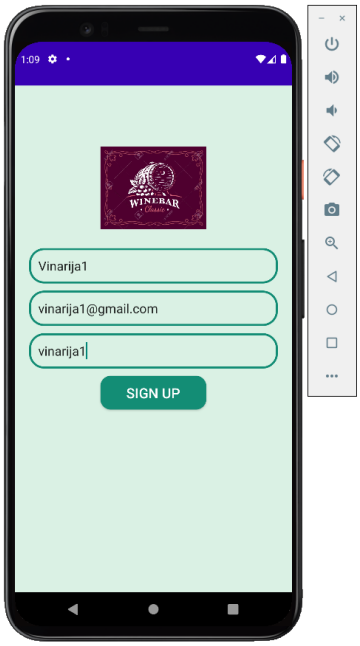
Pri pokretanju aplikacije pokrenut je početni zaslon *LogInActivityja* koji omogućuju prijavu korisnika. Budući da korisnik nije registriran, registraciju započinje pritiskom na gumb „SIGN UP“. Početni zaslon aplikacije prikazan je na slici 4.1.



Sl. 4.1 Početni zaslon aplikacije

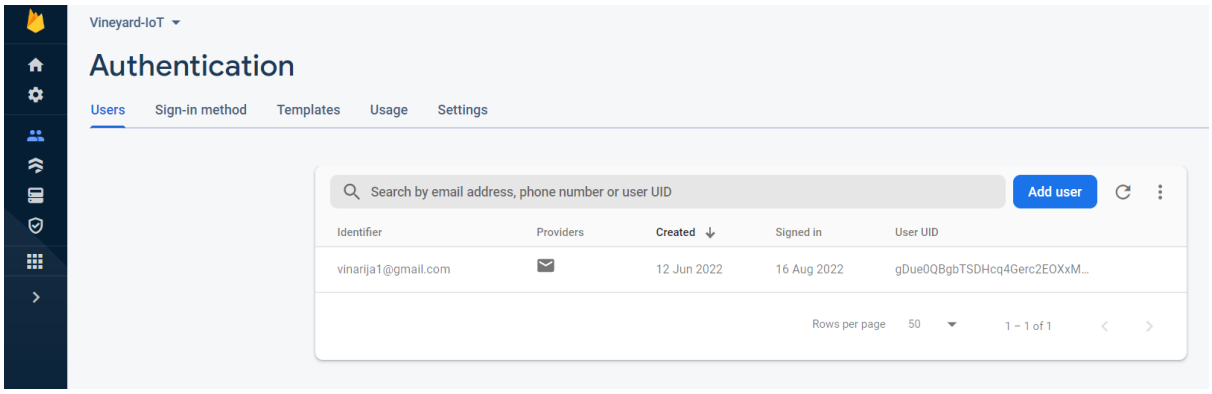
4.2 Registracija korisnika

Pritiskom gumba „SIGN UP“ otvara se novi zaslon *SignUpActivityja* koji korisniku služi za registraciju. Slika 4.2 prikazuje ispunjeni obrazac za registraciju s primjerom naziva vinarije, email adrese i lozinke.



Sl. 4.2 Ispunjeni obrazac za registraciju

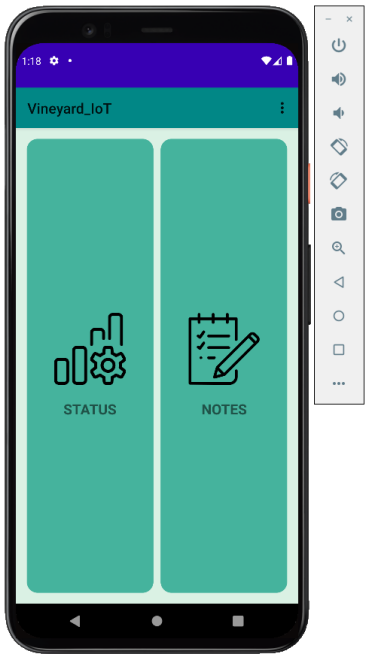
Registracija se izvršava pritiskom gumba „SIGN UP“. Slika 4.3 prikazuje tablicu korisnika registriranih u *Firebase Authentication* tablici korisnika nakon registracije prvog korisnika.



Sl. 4.3 *Firebase Authentication* tablica s registriranim korisnicima

4.3 Odabir aktivnosti

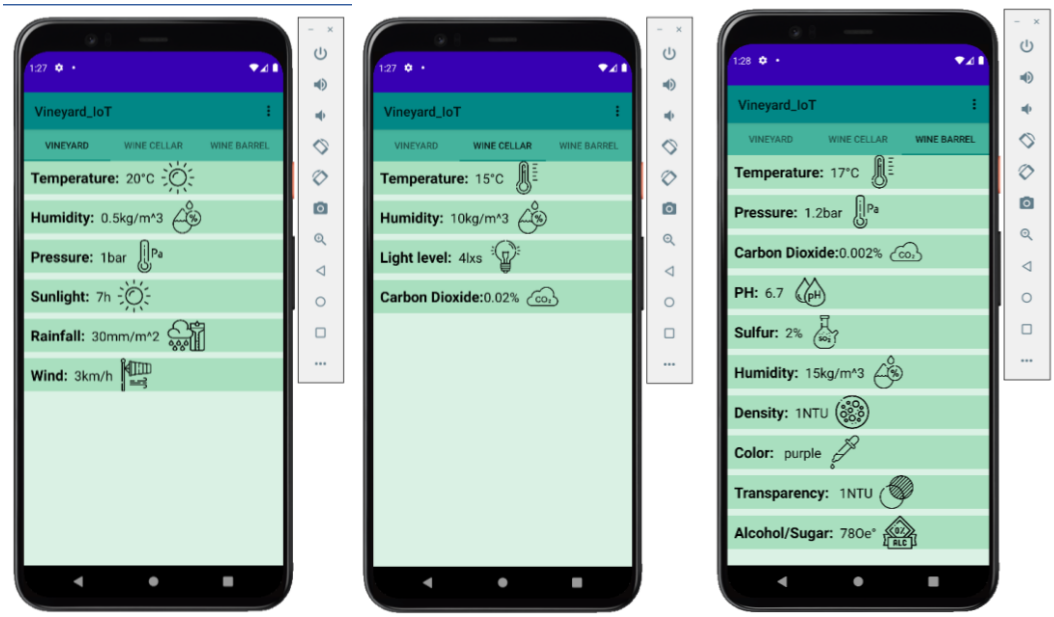
Nakon pritiska na gumb „SIGN UP“ otvara se zaslon *SelectActivityja* kako je prikazano na slici 4.4. Odabirom na lijevi dio zaslona s natpisom „STATUS“ i simboličnom ikonom otvara se *MainActivity* s popisom parametara i njihovih vrijednosti. Pritiskom na desni dio zaslona s natpisom „NOTES“ otvara se *NotesActivity* s mogućnosti zapisivanja bilješki.



Sl. 4.4 Zaslon *SelectActivityja*

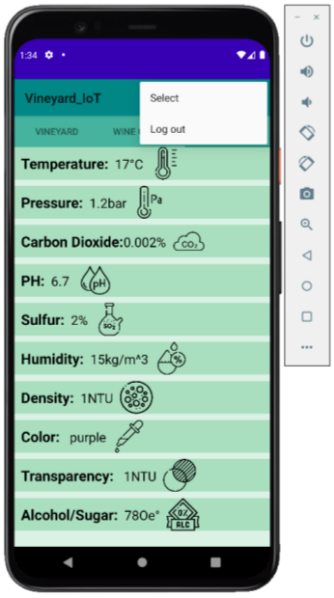
4.4 Očitavanje parametara

Pritiskom na lijevi dio zaslona („STATUS“) otvara nam se *MainActivity* u kojem se nalazi popis parametara s obzirom namjesto očitanja. Slika 4.5 prikazuje popis parametara koji se izmjenjuje klizanjem preko zaslona prema desno otvaranjem sljedećeg fragmenta.



Sl. 4.5 Izmjena fragmenata klizanjem preko zaslona

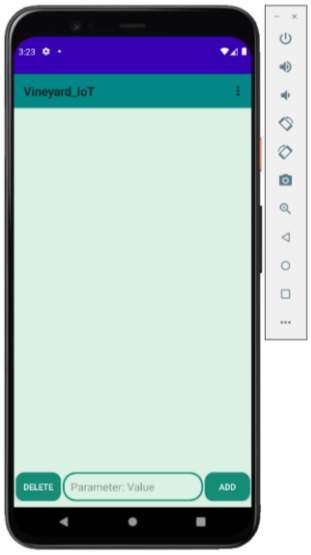
Pritiskom na tri točkice koje se nalaze u gornjem desnom uglu otvora se izbornik s mogućnosti odjave korisnika ili povratka na *SelectActivity* s ponovnom mogućnosti odabira aktivnosti (Slika 4.6).



Sl. 4.6 Izbornik

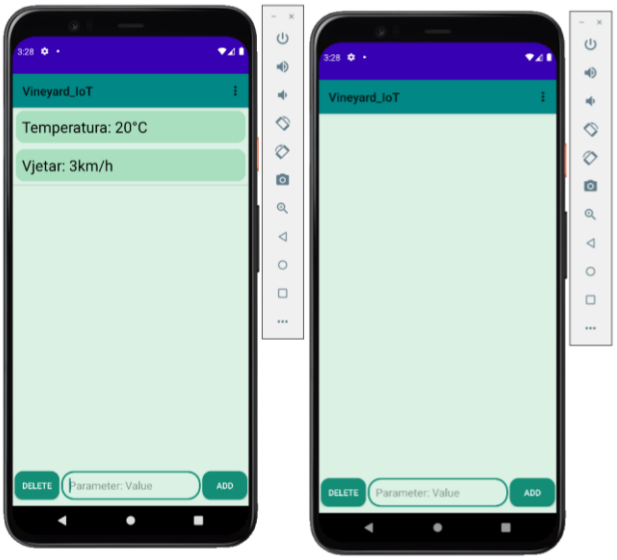
4.5 Zapisivanje bilješki

Povratkom na zaslon *SelectActivityja* te pritiskom na desni dio zaslona s natpisom „NOTES“ otvara se zaslon *NotesActivityja* koji je prikazan na slici 4.7.



Sl. 4.7 Zaslon za zapisivanje bilješki

Upisivanjem bilješke i pritiskom na gumb „ADD“ dodaju se bilješke te se prikazuju redoslijedom dodavanja odozgor prema dolje od vrha zaslona. Slika 4.8 prikazuje zaslon nakon dodavanja nekoliko bilješki te zaslon nakon pritiska gumba „DELETE“ kojim su obrisane sve bilješke.



Sl. 4.8 Zaslon nakon dodavanja i brisanja bilješki

5. ZAKLJUČAK

U ovom radu napravljena je aplikacija za pomoć vinogradarima pri proizvodnji vina. Svojim mogućnostima omogućuje lakše praćenje procesa sazrijevanja grožđa u vinogradu kao i procesa dozrijevanja vina u vinskom podrumu i bačvi. Aplikacija nudi registraciju i prijavu već registriranih korisnika pomoću *Firebase Authentication* usluge. Korisnik prijavom ima mogućnost očitavanja parametara čije se vrijednosti nalaze u *Cloud Firestore* bazi podataka. Osim očitavanja parametara, korisnik ima mogućnost zapisivanja bilješki koje mu pomažu u svakodnevnim poslovima u vinogradu i vinariji.

Aplikacija je testirana u Android virtualnom uređaju te dokazano obavlja sve potrebne funkcionalnosti. U narednim verzijama aplikacije moguće je dodati obavijesti koje bi korisnika upozorile ukoliko dođe do nagle promijene parametara te uređivanje baze podataka za obradu bilješki.

LITERATURA

[1] homewine.eu, "HomeWine Premium", homewine.eu, 2015. Dostupno: https://play.google.com/store/apps/details?id=pl.iitis.homewine.premium&hl=bs&gl=US

[2] Brew Ventures, "EnoFile - Home Winemaking", Brew Ventures, 2018. Dostupno: https://play.google.com/store/apps/details?id=ventures.brew.enofile&hl=hr&gl=US.

[3] M. Gašpar i A. Karačić, "Podizanje vinograda sa zaštitom vinove loze", Mostar, Federalni agromediteranski zavod Mostar, 2011..

[4] Agroklub, "Temperatura u vinskom podrumu", 11. 04. 2010., Dostupno: https://www.agroklub.com/vinogradarstvo/temperatura-u-vinskom-podrumu/2865/.

[5] "Wine Club Directory", 2021., Dostupno: https://hr.wineclubdirectory.net/wine-cellar-how-design-perfect-one-for-your-wine-collection.

[6] Agroklub, "Dozrijevanje vina u podrumu za bolji okus", 14. 12. 2018., Dostupno: https://www.agroklub.com/vinogradarstvo/dozrijevanje-vina-u-podrumu-za-bolji-okus/47222/.

[7] D. Tomas i D. Kolovrat, "Priručnik za proizvodnju vina", Mostar, Federalni agromediteranski zavod Mostar, 2011..

[8] "Android Studio", 2022. , Dostupno: https://en.wikipedia.org/wiki/Android_Studio.

[9] M. Heller, "What is Kotlin? The Java alternative explained," 23. 03. 2020., Dostupno: https://www.infoworld.com/article/3224868/what-is-kotlin-the-java-alternative-explained.html.

[10] Google, "Firebase" Google, Dostupno: https://firebase.google.com/products-build.

[11] Fakultet elektrotehnike, računarstva i informacijskih tehnologija, Osnove razvoja Web i mobilnih aplikacija, LV8, Osijek, 2021..

[12] Google, "onCreateOptionsMenu", Google , Dostupno: https://developer.android.com/reference/android/app/Activity#onCreateOptionsMenu(android.view.Menu).

SAŽETAK

Cilj ovog završnog rada je izrada Android mobilne aplikacije koja vinaru omogućuje praćenje parametara bitnih za proizvodnju i obradu vina. Parametri su mjereni u vinogradu, vinskom podrumu i vinskoj bačvi. Korisnik ima mogućnost očitanja svih parametara spremljenih u *Cloud Firestore* bazi podataka. Za izradu aplikacije korišteno je razvojno okruženje *Android Studio*. Opisnim jezikom XML definiran je izgled aplikacije, a programskim jezikom *Kotlin* definirana je funkcionalnost aplikacije. Aplikacija je testirana te je utvrđeno kako je korisniku omogućeno pokretanje aplikacije te unos korisničke email adrese i lozinke, ali i ostale zadane mogućnosti. U budućnosti moguće su nadogradnje koje obuhvaćaju daljnji rad s parametrima te obrada podataka u svrhu još lakšeg i točnijeg procesa praćenja proizvodnje vina.

Ključne riječi: Android Studio, Firebase, Kotlin, mobilna aplikacija, vinarstvo

ABSTRACT

The aim of this final work is the development of an Android mobile application that enables the winemaker monitoring the parameters important for the production and processing of wine. The parameters are measured in the vineyard, wine cellar and wine barrel. The user has the ability to read all parameters saved in the *Cloud Firestore* database. The *Android Studio* development environment was used to create the application. The interface of the application is defined with the description language XML, and the functionality of the application is defined with the programming language Kotlin. The application was tested and it was determined that the user is enabled to launch the application and enter the user's email address and password, as well as other default options. In the future, there would be possibility for upgrades that include further work with parameters and data processing for the purpose of an even easier and more accurate process of monitoring wine production

Key words: Android Studio, Firebase, Kotlin, mobile application, winemaking

ŽIVOTOPIS

Mateo Tokić rođen je 25. rujna 2000. godine u Zagrebu. 2015. godine završio je Osnovnu školu „Ljudevit Gaj“ u Lužanima. Nakon završene osnovne škole upisuje Klasičnu gimnaziju fra Marijana Lanosovića s pravom javnosti u Slavonskom Brodu koju završava 2019.godine. Iste godine upisuje preddiplomski sveučilišni smjer elektrotehnike na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija koji i danas pohađa.

PRILOZI (na USB-u)

Prilog 1. Dokument završnog rada

Prilog 2. PDF završnog rada

Prilog 3. Programski projekt aplikacije