IZRADA ANDROID APLIKACIJE ZA PRODAJU OPG PROIZVODA

Čotić, Kristijan

SVEUČILIŠTE U SPLITU

SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Specijalistički diplomski stručni studij Informacijske tehnologije

KRISTIJAN ČOTIĆ

Z A V R Š N I R A D

IZDRADA ANDROID APLIKACIJE ZA PRODAJU OPG PROIZVODA

Split, lipanj 2019.

SVEUČILIŠTE U SPLITU
SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Specijalistički diplomski stručni studij Informacijske tehnologije

Predmet: Mobilne tehnologije

Z A V R Š N I R A D

Kandidat: Kristijan Čotić

Naslov rada: Izrada Android aplikacije za prodaju OPG proizvoda

Mentor: Marina Rodić, predavač

Split, lipanj 2019.

Sadržaj

SAŽETAK

Cilj rada je izrada aplikacije za kupovinu domaćih proizvoda koja bi olakšala poljoprivrednicima plasiranje svojih proizvoda na tržište, a ujedno i kupcima kako bi pomoću svojih mobilnih uređaja došli do njih.

Koristeći ovu aplikaciju proizvođač može dodavati nove kategorije proizvoda, nove proizvode, zaprimati narudžbe od kupaca te upravljati tim istim narudžbama, dok kupac može pregledavati kategorije proizvoda, proizvode, kreirati narudžbe, pregledavati vlastite narudžbe te zaprimati račune putem elektroničke pošte.

Aplikacija je namijenjena mobilnim uređajima koji rade na operacijskom sustavu Android. Za izradu rada koristi se Android studio koji je Google-ova razvojna okolina za izradu aplikacija baziranih na Androidu, usluga za pohranu podataka Firebase koja je također Google-ov proizvod i koristi JSON (eng. *JavaScript Object Notation*) format za spremanje podataka te Adobe Acrobat pro koji se koristi za izradu PDF formi koje je moguće naknadno popunjavati.

Cilj aplikacije je olakšati plasiranje proizvoda proizvođačima, a kupcima pojednostavniti da što lakše dođu do kvalitetnih domaćih proizvoda.

Ključne riječi: Firebase, JSON, mobilni uređaji, operacijski sustav Android

SUMMARY

Android application development for selling family farm products

The aim of this work is to create an application for buying domestic products that would make it easier for farmers to place their products on the market and at the same time for customers to access it from their mobile devices.

With this application, the manufacturer can add new product categories, new products, receive customer orders, and manage these same orders, while the customer can view product categories, products, create orders, view their own orders, and receive invoices via e-mail.

The app is intended for Android mobile devices due to the widespread availability of this operating system. For making this work we used Android studio that is Google's development platform for Android-based applications, Firebase data storage service, which is also a Google product, uses the JSON format to save data and Adobe Acrobat Pro, which is used to create PDF forms that can be filled out later.

By creating this application, manufacturers have easier way to break into the market and place their products, and customers can on faster and easier way access to quality domestic products.

Keywords: Firebase, JSON, mobile devices, operating system Android

1. Uvod

OPG [1] (obiteljsko poljoprivredno gospodarstvo) obavlja poljoprivredne i ostale dopunske djelatnosti. Primarna djelatnost OPG-ova je proizvodnja poljoprivrednih proizvoda i njihova prodaja.

Cilj rada je izrada mobilne aplikacije koja će vlasnicima OPG-ova omogućiti novi način plasiranja svojih proizvoda na tržište za razliku od dosadašnje prodaje na tržnicama. Također sa druge strane kupci će umjesto odlaska na tržnicu do domaćih proizvoda moći doći u nekoliko klikova. Razlog odabira izrade mobilne aplikacije umjesto Web aplikacije je taj što su vlasnici OPG-ova (prodavači) zbog prirode posla konstantno u pokretu i nemaju previše vremena sjediti za računalom, slična stvar je i sa kupcima.

OPG internet trgovina se sastoji od dvije mobilne aplikacije prilagođene za android mobilne uređaje. Aplikacija za prodavača i aplikacija za kupca. Aplikacije su napisane pomoću Android studio razvojne okoline koja koristi Java programski jezik, a s njom u paketu dolazi i Android SDK. Za spremanje podataka odnosno informacija o kategorijama, proizvodima i narudžbama koristi se Firebase usluga koja nudi mogućnost pohrane podataka.

U drugom poglavlju su opisane tehnologije koje su se koristile prilikom izrade aplikacija. Treće poglavlje opisuje funkcionalnosti aplikacije za prodavača, aktivnosti (eng. *activity*) i izgled korisničkog sučelja. U četvrtom poglavlju su opisane funkcionalnosti aplikacije za kupca, aktivnosti i izgled korisničkog sučelja. U petom poglavlju će se opisati neke dodatne funkcionalnosti i struktura baze podataka. U šestom poglavlju nalazi se zaključak.

2. Tehnologije

2.1. Android

Operacijski sustav Android je operacijski sustav koji je razvio Google. Na početku je to bila zasebna tvrtka (startup) kojoj je namjera bila razviti napredni operacijski sustav za digitalne kamere. Google je vidio potencijal u toj kompaniji i kupio je 2005. godine. Nakon preuzimanja kompanije zaključili su kako je tržište digitalnih fotoaparata ograničeno i odlučili su prenamijeniti sustav u operacijski sustav za mobilne uređaje. Prva verzija sustava predstavljena je 2007. godine i od tada pa do danas njegov udio na tržištu operacijskih sustava za mobilne uređaje nezaustavljivo raste. Danas njegov udio na tržištu iznosi 75.27 % [8].

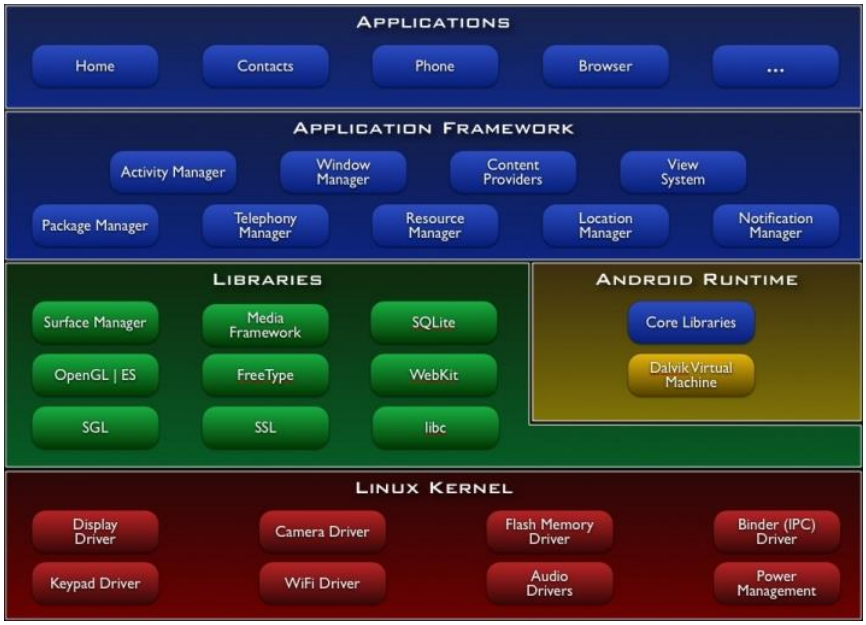
2.1.1. Operacijski sustav Android

Operacijski sustav Android [2] je baziran na prilagođenoj verziji Linux operacijskog sustava i otvorenog je koda. Prilagođen je uređajima sa zaslonom osjetljivim na dodir poput mobitela i tableta, a u novije vrijeme se proširio na pametne satove, televizore, automobile i ostale kućanske uređaje.

Kao što smo prethodno napisali operacijski sustav Android je otvorenog koda što znači da svaki proizvođač može uzeti njegov kod i prilagoditi ga prema svojim potrebama. Iako je otvoreni pristup velika prednost operacijskog sustava Android jer omogućava da svi sudjeluju u njegovom daljnjem razvoju to je ujedno i njegova mana jer ga različiti proizvođači prilagođavaju i mijenjaju prema svojim potrebama. Posljedica toga je da proizvođač prilagodi određenu verziju sustava jednom uređaju te izlaskom novog uređaja prestane razvijati podršku za prethodni što dovodi do tog da nove verzije aplikacija ne rade na tim uređajima, a i ti uređaji postaju izloženi sigurnosnim rizicima. Također nerijetko se događa da aplikacije različito reagiraju na uređajima raznih proizvođača.

Arhitektura operacijskog sustava Android se sastoji od nekoliko slojeva kao što je vidljivo na slici Slika 1. Na dnu se nalazi Linux jezgra (eng. *kernel*) verzije 2.6. Zatim slijedi Android Runtime koji se sastoji od Dalvik virtualnog sloja i jezgrenih biblioteka (eng. *library*). Sljedeći sloj je aplikacijski okvir koji se sastoji od ponovno iskoristivih komponenata koje pomažu u razvoju aplikacija poput elemenata prikaza (eng. *Views*), pružatelja sadržaja (eng. *Content Providers*), upravitelja resursa (eng. *Resource Manager*),

upravitelja obavijesti (eng. *Notification Manager*) i upravitelja aktivnosti (eng. *Activity Manager*). Zadnji sloj je aplikacijski sloj koji se sastoji od ugrađenih i naknadno preuzetih i instaliranih aplikacija. Slojevi nisu jasno odvojeni već se isprepleću.



Slika 1: Arhitektura Android sustava [9]

2.1.2. Android studio

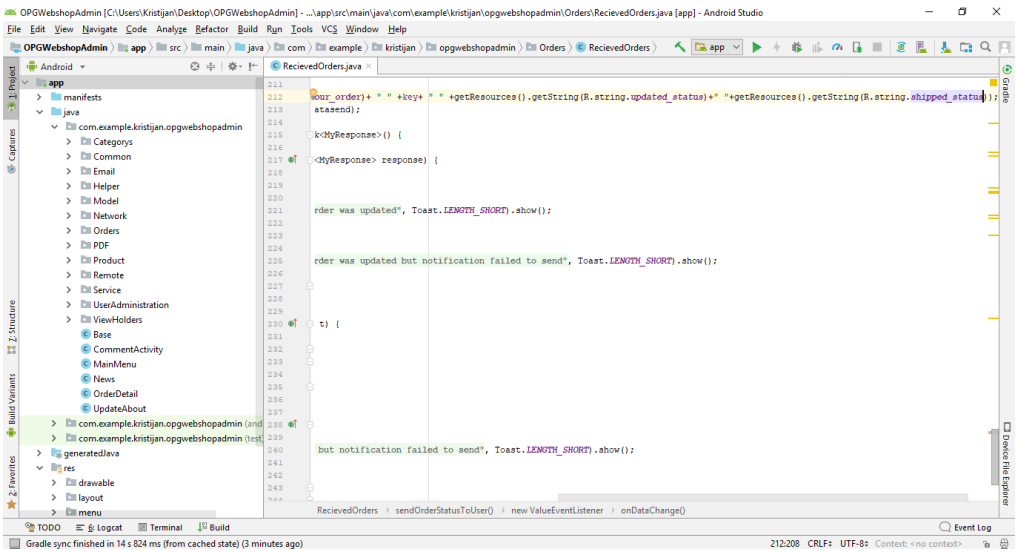
Android – studio je programsko rješenje razvijeno 2014. godine za izradu Android aplikacija. Prije njegova nastanka za izradu su se koristila razna razvojna okruženja u koja je trebalo uključiti ADK (Android Development Tools) od kojih je najpopularniji bio Eclipse. U vrlo kratkom vremenu je istisnuo ostala razvojna okruženja te postao najpopularniji alat za razvoj Android aplikacija. Za razvoj aplikacija osim samog Programa potrebno je instalirati i JDK (Java Development Kit). Osim izrade aplikacija u Java programskom jeziku nudi i podršku za Kotlin programski jezik.

Prednost Android studija je niz dodatnih mogućnosti poput:

- Google Wear- izrada aplikacija namijenjenih pametnim satovima
- Uređivač prijevoda – omogućava lagano prevođenje teksta koji se prikazuje korisniku na različite jezike
- Uređivač sučelja (eng. *layout*) – nudi širok izbor elemenata koje možemo rasporedit po korisničkom sučelju
- Uređivač vektora – omogućava stvaranje ikona za ekrane različite gustoće piksela

- Podrška za Firebase i Cloud platformu – omogućava nam da u projekt uključimo razne mogućnosti poput usluge za pohranu podataka, autentifikacije (eng. *authentification*), obavijesti…
- Gradle build alat – za prevođenje koda i komprimiranje u aplikacijsku datoteku. Također omogućava da se u projekt uključe vanjske biblioteke.

Sučelje android studija na slici Slika 2 možemo podijeliti u nekoliko dijelova. Najveći dio prozora zauzima WYSIWYG editor koji sadržava razne izbornike i uređivač koda, sa desne strane se nalazi pregled našeg projekta sa svim mapama i datotekama, na vrhu se nalazi alatna traka sa najvažnijim funkcijama (spremanje projekta otklanjanje pogrešaka, pokretanje projekta…), a u podnožju se nalaze razni skočni izbornici koji prikazuju informacije prilikom pokretanje projekta i informacije o greškama.



Slika 2: Android Studio

2.2. Firebase

Firebase [3] je platforma za razvoj mobilnih i Web aplikacija osnovana 2011. godine, a Google ju je kupio 2014. godine. Integrirana je u razne Google-ove usluge uključujući i Android Studio. Trenutno uključuje 18 proizvoda poput usluge za pohranu podataka, pohrane slika i dokumenata, slanja udaljenih poruka, autentifikacije i izvješća o greškama.

Alati koji su korišteni tijekom izrade aplikacije:

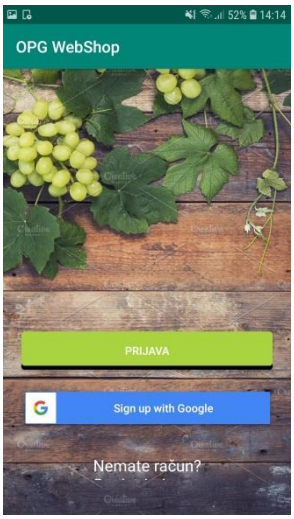
- **Realtime Database** – omogućava spremanje brisanje i ažurirane podataka u stvarnom vremenu. Podaci se spremaju u JSON formatu u obliku objekta koji sadrži kolekciju podataka u formatu parova ključ-vrijednost. Velika prednost je što ne postoji potreba za nabavkom i pokretanje vlastite poslužiteljske opreme. Jedna od većih mana Realtime baze su upiti i sortiranje.
- **Firebase Auth** – je usluga koja omogućava autentifikaciju korisnika koristeći kod samo na klijentskoj strani. Nudi razne metode autentifikacije putem društvenih mreža (Facebook, Google, GitHub) i upravljanje i autentifikaciju korisnika putem adrese elektroničke pošte i lozinke. Ova metoda omogućava provjeru valjanosti korisnikove adrese i resetiranje lozinke putem elektroničke pošte.
- **Firebase Storage** – omogućava sigurno spremanje i preuzimanje raznih datoteka poput slika, videa, zvuka i ostalih datoteka.
- **Firebase Cloud Messaging** – višeplatformsko rješenje koje omogućuje slanje poruka i obavijesti operacijskog sustava Android, IOS i Web aplikacijama. Uključuje nekoliko API-a (eng. *Application programming interface*) popit slanja obavijesti, slanja grupnih obavijesti , uključivanje i isključivanje grupnih obavijesti.

3. Sučelje i funkcionalnosti aplikacije za kupce

U ovom dijelu se opisuje izrada aplikacije namijenjene kupcima. Prikazat će se izgled korisničkog sučelja i dijelovi programskog koda koji omogućuju najvažnije funkcionalnosti. Ako je jedna funkcionalnost pojašnjena u jednoj aktivnosti, u sljedećoj se neće posebno pojašnjavati. Podjela je napravljena po aktivnostima od registracije korisnika u sustav do zaključivanja narudžbe.

3.1. Početni zaslon

Na početnom zaslonu prikazanom na slici Slika 3 se nalazi botun za prijavu korisnika, botun za prijavu putem Google autentifikacije i veza na aktivnost za kreiranje korisničkog računa. Početni zaslon će se pojaviti samo ako korisnik nije prethodno prijavljen, u suprotnom korisniku se prikazuje glavni izbornik sa kategorijama proizvoda.



Slika 3: Početni zaslon

Od funkcionalnosti će se istaknuti prijava korisnika pomoću Google autentifikacije [4]. Unutar Gradle skripte potrebno je dodati biblioteku za korištenje ove metode prijave. Instanciranjem (eng. *instance*) klase GoogleSignInOptions i korištenjem metode requestIdToken() koja pripada toj klasi obavlja se konfiguracija Google-ova API-a za prijavu (Ispis 1). Konfiguracija API-a je nužna jer se taj objekt koristi za instanciranje klase koja je zadužena za pokretanje aktivnosti za prijavu.

```
GoogleSignInOptions gso = new
GoogleSignInOptions.Builder(GoogleSignInOptions.DEFAULT_SIGN_IN)
        .requestIdToken(getString(R.string.default_web_client_id))
        .requestEmail()
        .build();
```

Ispis 1: Konfiguracija API-a za prijavu

Povratna vrijednost aktivnosti za prijavu odnosno dobiveni račun korisnika se koristi unutar metode firebaseAuthWithGoogle(). Ta metoda radi zamjenu Google računa za Firebase vjerodajnice (eng. *credentials*) kako bi se moglo prijaviti pomoću Firebase autentifikacije što je prikazano u ispisu Ispis 2.

```
private void firebaseAuthWithGoogle(GoogleSignInAccount account) {
    AuthCredential credential =
GoogleAuthProvider.getCredential(account.getIdToken(), null);
    mAuth.signInWithCredential(credential)
            .addOnCompleteListener(this, new
OnCompleteListener<AuthResult>() {
                @Override
                public void onComplete(@NonNull Task<AuthResult> task) {
                    if (task.isSuccessful())
                    Intent menu2 =new
Intent(MainActivity.this,MenuHome.class);
                        startActivity(menu2);
                        FirebaseUser user = mAuth.getCurrentUser();
                    }
                }
            });
}
```

Ispis 2: Zamjena za Firebase vjerodajnice

3.2. Prijava i registracija korisnika

Prijava korisnika prikazana na slici Slika 4 se sastoji od tekstualnih polja za unos korisničke adrese elektroničke pošte i lozinke, veze na aktivnost u slučaju zaboravljene lozinke i botuna za prijavu.

Prilikom prijave korisnik mora popuniti sva polja. U suprotnom korisniku se na zaslon ispisuje poruka da adresa elektroničke pošte ili lozinka nisu uneseni. Također pritiskom na botun Prijava u slučaju krivo unesenih podataka korisnika se obavještava da su lozinka ili adresa elektroničke pošte netočni. Kada korisnik unese ispravne podatke ispisuje se poruka o uspješnoj prijavi i korisnika se prebacuje na sljedeću aktivnost odnosno glavni izbornik.

Slika 4: Sučelje prijave korisnika

Ako korisnik nema korisnički račun potrebno ga je kreirati. Korisničko sučelje je slično sučelju za prijavu. Razlike su u tome što ako korisnik unese prekratku lozinku ne dozvoljava mu se prijava te se ispiše poruka kako je unesena lozinka prekratka i korisnik nakon uspješne registracije dobije elektroničku poštu za potvrdu adrese kojom se prijavio.

3.2.1. Povezivanje GUI elemenata i njihovo ponašanje

GUI (grafičko korisničko sučelje) elemente i njihov izgled se kreira i oblikujemo u uređivaču sučelja (eng. *Layout editor*) na način da definiramo njegove atribute. Prikaz kreiranja botuna za prijavu korisnika je prikazan u ispisu Ispis 3.

```
<com.rengwuxian.materialedittext.MaterialEditText
    android:id="@+id/txt_password_login"
    android:layout_width="314dp"
    android:layout_height="wrap_content"
    android:hint="@string/password_hint"
    android:inputType="textPassword"
    android:textSize="32sp"
    app:met_floatingLabel="highlight"
    app:met_minCharacters="6"
 app:met_primaryColor="@color/Edit_txt_focus"
    app:met_singleLineEllipsis="true"
    app:met_textColor="@android:color/white"
    />
```

Ispis 3: Kreiranje botuna

Za pristup tim elementima potrebno je deklarirati objekt odgovarajuće klase i povezati ga s grafičkim elementom (Ispis 4).

```
FButton Login_Btn;
Login_Btn=(FButton) findViewById(R.id.Login_Button_on);
```

Ispis 4: Povezivanje botuna

Kako bi se definiralo ponašanje određenog elementa, odnosno u ovom slučaju što će se dogoditi pritiskom na botun Prijava, koristi se sučelje: setOnClickListener koje u sebi sadrži metodu onClick. Unutar te metode se definiraju akcije koje će se izvršiti.

3.2.2. Proces prijave korisnika

Za prijavu korisnika se koristi Firebase Auth koji je dio Firebase-a. Kako bi se mogla koristiti ova funkcionalnost unutar Gradle skripte potrebno je uključiti Firebase auth biblioteku.

Za implementaciju same funkcionalnosti kreira se objekt klase FirebaseAuth. Njegova metoda signInWithEmailAndPassword se koristi za proces prijave na način da joj se proslijedi adresa elektroničke pošte i lozinka koji se unose u tekstualna polja što je vidljivo iz ispisa Ispis 5.

```
auth.signInWithEmailAndPassword(email, password)
.addOnCompleteListener(Login.this, new OnCompleteListener<AuthResult>()
```

Ispis 5: Povezivanje botuna

Metode onComplete dohvaća odgovor poslužitelja o uspješnosti autentifikacije U slučaju pozitivnog odgovora uključena je dodatna provjera koja provjerava da li je korisnik potvrdio adresu svoje elektroničke pošte. Ako je adresa potvrđena, korisniku se dozvoljava prijava i preusmjerava ga se na sljedeću aktivnost, a u suprotnom korisnik se ne može prijaviti. Provjera da li je adresa potvrđena je prikazana u ispisu Ispis 6.

```
if (auth.getCurrentUser().isEmailVerified()) {
    mdialog.dismiss();
    Intent menu = new Intent(Login.this, MenuHome.class);
    startActivity(menu);
    finish();
}
```

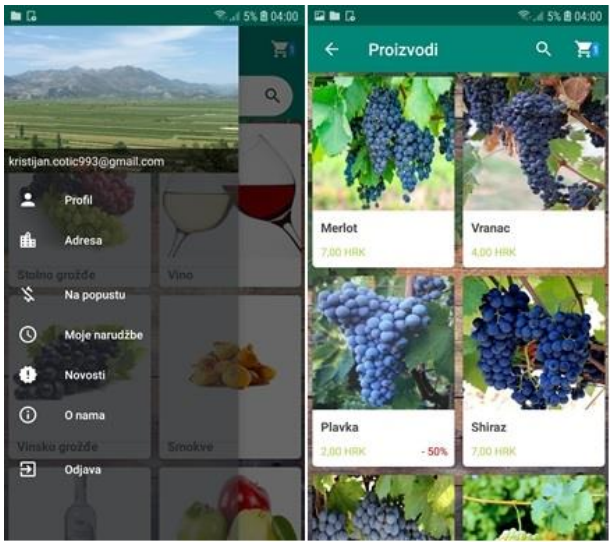
Ispis 6: Provjera potvrde adrese elektroničke pošte

Implementacija registracije korisnika je slična prijavi, iako postoje neke manje razlike. Ključna razlika je da umjesto metode signInWithEmailAndPassword() koristi metoda createUserWithEmailAndPassword().

3.3. Glavni izbornik i proizvodi

Sučelje glavnog izbornika se sastoji od pregleda kategorija, navigacijskog izbornika i tražilice. Izbornik se otvara dodirom na botun u obliku ljestava koji se nalazi na lijevoj strani navigacijske trake u kojoj se nalazi i košarica ako korisnik ima dodan barem jedan artikl. Pomoću navigacijskog izbornika možemo pristupiti raznim aktivnostima poput pregleda korisničkog profila, adrese, korisnikovih narudžbi, informacija o OPG-u i mogućnosti odjave. Klikom na tražilicu se otvara nova aktivnost unutar koje je moguće pretraživanje proizvoda.

Korisnik može pregledavati razne kategorije proizvoda, te se klikom na određenu kategoriju otvara aktivnost sa proizvodima koji spadaju u tu kategoriju. U prethodnu kategoriju se možemo vratiti klikom na strelicu u navigacijskoj traci. Slika 5 prikazuje Početni izbornik i aktivnost proizvoda.



Slika 5: Glavni izbornik i proizvodi

3.3.1. Implementacija kategorija

Podaci o određenoj kategoriji se čitaju iz Firebase usluge za pohranu podataka. Kako bi se koristila ova usluga potrebno je instancirati dvije klase FirebaseDatabase i DatabaseReference. Ove klase se koriste za kreiranje instance baze i reference na određene podatke unutar baze.

```
database=FirebaseDatabase.getInstance();
category= database.getReference("category");
```

Ispis 7: Instanca baze i referenca na podatke

Za prikaz kategorija se koristi RecyclerView koji omogućava kreiranje skrolajuće (eng. *scrolling list*) liste elemenata. RecyclerView se puni sa pogledima (eng. *Views*) koji su pruženi od strane layout manager-a. U ovom slučaju se koristi GridLayoutManager pomoću kojega se elementi mogu prikazati u obliku mreže. Prikazano u ispisu Ispis 8.

```
recycler_menu=(RecyclerView) findViewById(R.id.category_recycler);
recycler_menu.setHasFixedSize(true);
recycler_menu.setLayoutManager(new GridLayoutManager(this,2));
```

Ispis 8: GridLayoutManager

Pogledi unutar RecyclerView-*a* su predstavljeni pomoću view holder objekta i svaki od tih objekata je zadužen za predstavljanje jednog elementa unutar RecyclerView-a. Tim objektima se upravlja pomoću adaptera. U ovom slučaju se ne koristi vlastiti adapter već FirebaseUI biblioteka. Unutar te biblioteke se nalazi klasa FirebaseRecyclerAdapter. Pomoću nje se može automatski napraviti upit prema bazi podataka i dobivene podatke prikazati na zaslonu u obliku liste. Za rad s tom klasom potrebno je kreirati model podataka i proširiti ViewHolder klasu koji se zajedno sa upitom na određene podatke u bazi prosljeđuju, konstruktoru klase FirebaseRecyclerAdapter-a.

Model podataka predstavlja podatke u bazi. Koristeći model podataka na lakši način se može mapirati (eng. *dana mapping*) podatke na određenu lokaciju u bazi i pročitane podatke iz baze spremiti u obliku objekata. Ispis 9 prikazuje model podataka za kategorije.

```
public class Category {
    private String image;
    private  String name;

    public Category() {
    }

    public Category(String image, String name) {
        this.image = image;
        this.name = name;
    }

    public String getImage() {
        return image;
    }

    public void setImage(String image) {
        this.image = image;
    }

 }
```

Ispis 9: Model podataka kategorije

ViewHolder klasa opisuje pogled na određenu stavku unutar RecyclerView-a. Unutar nje se povezuju grafički elementi stavki i definira ponašanje kao što je mogućnost klika na stavku. Glavna prednost korištenja ViewHolder je smanjen broj poziva prema findViewById() metodi. U ispisu Ispis 10 je prikazan konstruktor i metode prilagođene ViewHolder klase.

```
public CategoryHolder(@NonNull View itemView) {
    super(itemView);
    cat_text=(TextView) itemView.findViewById(R.id.image_name);
    cat_img=(ImageView) itemView.findViewById(R.id.image_model);
    itemView.setOnClickListener(this);
}
public void setItemClickListener(ItemClickListener itemClickListener) {
    this.itemClickListener = itemClickListener;
}
```

Ispis 10: Konstruktor i metoda klase

Za sam prikaz podataka na poglede unutar RecyclerView-a potrebno je pregaziti ponašanje metode populateViewHolder()koja pripada klasi FirebaseRecyclerAdapter vlastitom implementacijom.

U ispisu Ispis 11 je prikazano postavljanje imena i slike kategorije na svaki pojedini pogled unutar RecyclerView-a na način da se u polja za prikaz unutar pogleda učitavaju podaci iz objekta koji predstavljaju određeni model podataka.

```
protected void populateViewHolder(CategoryHolder viewHolder, Category
model, int position) {
    viewHolder.cat_text.setText(model.getName());
    String imgUrl = model.getImage();
Glide.with(getBaseContext())
.load(imgUrl).thumbnail(0.5f).into(viewHolder.cat_img);
```

Ispis 11: Prikaz stavki u RecyclerView-u

3.3.2. Aktivnost proizvoda

Aktivnost proizvoda nije potrebno posebno opisivati jer su sve funkcionalnosti implementirane na isti način. Jedina razlika je u prikazu i obračunavanju popusta. U slučaju da postoji popust na određeni proizvod pomoću metode setVisibility() koja služi za skrivanje ili prikaz određenog GUI elementa prikazuje se grafički element popusta. Prikazano u ispisu Ispis 12.

```
if (discoount > 0)
{
    viewHolder.product_discount.setVisibility(View.VISIBLE);
}
```

Ispis 12: Prikaz popusta

Obračunavanje popusta se radi unutar populateViewHolder metode što je prikazano u niže navedenom ispisu Ispis 13.

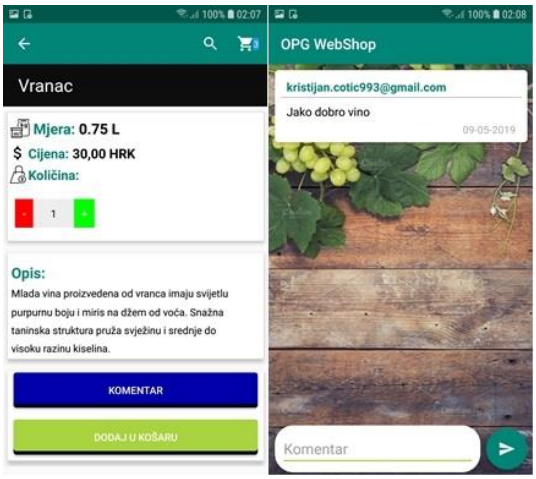
```
int discoount = Math.round((prod.getPrice() / 100.0f) *
prod.getDiscount());

int discounted_price=prod.getPrice()-discoount;
viewHolder.product_discount.setText(" - "+(String.valueOf(
prod.getDiscount())+"%"));
```

Ispis 13: Obračun popusta

3.4. Detalji proizvoda i komentari

Detalji proizvoda prikazuju određene informacije o proizvodu poput slike, naziva, cijene mjerne jedinice i popusta. Korisnik može sliku proizvoda sakriti unutar padajuće alatne trake. Osim informacija o proizvodu korisnik može odabrati željenu količinu proizvoda i pritiskom na botun „Dodaj u košaru“ dodati određeni proizvod u košaricu.

Pritiskom na botun „Komentari“ otvara se nova aktivnost gdje korisnik može vidjeti komentare za taj proizvod i dodati vlastiti komentar. Sučelje je prikazano na slici Slika 6.



Slika 6: Sučelje proizvoda i komentara

3.4.1. Prikaz informacija o proizvodu

Kako bi se prikazale informacije o određenom proizvodu kreira se instanca baze i referenca na lokaciju proizvoda. Korištenjem odgovarajućih metoda za dohvat i slušanje promjena na podacima, određeni podaci iz usluge za pohranu podataka se dohvaćaju i spremaju u objekt kojim su ti podaci predstavljeni. Podaci iz tog objekta se koriste za prikaz na GUI elementima.

Primjerice metoda child() klase DatabaseReference se koristi da bi se dohvatili podaci o određenom djetetu. Njenoj povratnoj vrijednosti se može postaviti metoda addListenerForSingleValueEvent() koja sluša promjene na podacima. Ona generira onDataChange()metodu pomoću koje se proizvod iz baze sprema u currentProd objekt. Ova metoda se poziva kod svake promjene nad podacima ili kod čitanja podataka. Na kraju se podaci iz objekta prikazuju na grafičkim elementima. Prikazano u ispisu Ispis 14.

```
public void onDataChange(@NonNull DataSnapshot dataSnapshot) {

    currentProd=dataSnapshot.getValue(Product.class);

    String imgUrl = currentProd.getImage();
Glide.with(getBaseContext()).load(imgUrl)
.thumbnail(0.5f).into(ProductImage);
ProductMesUnit.setText(currentProd.getMesUnit());
ProductDescription.setText(currentProd.getDescription());
    }
```

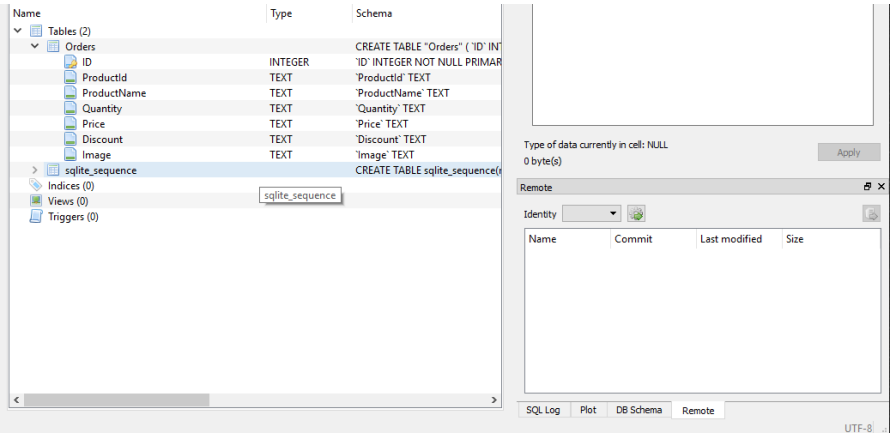
Ispis 14: Dohvaćanje i prikaz podataka

3.4.2. Dodavanje proizvoda u košaru

Kako bi se proizvod dodao u košaru koristi se lokalna baza podataka na uređaju. To se također moglo implementirati koristeći Firebase, ali kako korisnici često dodaju i brišu proizvode iz košarice radilo bi se previše upita prema poslužitelju pa se ovo činilo kao bolje rješenje.

Za implementaciju lokalne baze koristi se Android SQLiteAssetHelper biblioteka koja omogućava isporuku aplikacije sa unaprijed postavljenom bazom koja može i ne mora biti unaprijed popunjena sa podacima.

Prije svega je potrebno kreirati lokalnu bazu sa tablicom i odgovarajućim atributima i smjestiti je unutar asset mape. Prikazano na slici Slika 7.



Slika 7: Lokalna baza

Da bi se mogle implementirati metode za pisanje, čitanje, brisanje svih podataka i brisanje pojedinog podatka potrebno je kreirati vlastitu klasu na način da se proširi SQLiteAssetHelper klasa.

```
public void addToCart(Order order)
{
    SQLiteDatabase db = getReadableDatabase();
    String query = String.format
("INSERT INTO Orders(ProductId,ProductName,Quantity,Price,Discount,Image)
VALUES('%s','%s','%s','%s','%s','%s')",
            order.getProductId(),
            order.getProductName(),
            order.getQuantity(),
            order.getPrice(),
            order.getDiscount(),
            order.getImage());

    db.execSQL(query);
}
```

Ispis 15: Metoda za dodavanje u lokalnu bazu

U ispisu Ispis 16 je prikazana metoda za dodavanje podataka u bazu. SQLiteDatabase db = getReadableDatabase()kreira ili otvara bazu podataka. Nakon toga se kreira upit u obliku stringa (eng. *string*) koji se formatira. Na kraju je potrebno izvršiti upit pomoću metode execSQL.

Dodavanje proizvoda u košaru se radi na način da se instancira Database klasa te joj unutar konstruktora proslijedimo Order objekt. Order objekt je model podataka koji sadrži podatke o ID-u proizvoda, imenu, količini ,cijeni, popustu i poveznicu na sliku proizvoda.

```
new Database(getBaseContext()).addToCart(new Order(
        product_id,
        currentProd.getName(),
        String.valueOf(quantityViewDefault.getQuantity()),
        String.valueOf(discounted_price),
        String.valueOf(currentProd.getDiscount()),
        String.valueOf(currentProd.getImage())
));
```

Ispis 16: Dodavanje proizvoda u košaru

3.4.3. Komentiranje proizvoda

Komentari su implementirani na način da se kreira model podataka za komentare koji se sastoji od samog komentara, ID proizvoda, datuma i adrese elektroničke pošte trenutnog korisnika. Za dobivanje trenutnog datuma koristimo klase Locale i Calendar.

Spremanje komentara u bazu se radi pomoću push() metode koja pripada klasi DatabaseReference tako da automatski generira ključ pod kojim će podaci iz newComment objekta biti spremljeni u bazu. Spremanje podataka u bazu je prikazano na ispisu Ispis 17.

```
newComment= new Comment(com.getText().toString(),date,product_id
,FirebaseAuth.getInstance().getCurrentUser().getEmail());
comment.push().setValue(newComment);
```

Ispis 17: Spremanje komentara u bazu

3.4.4. Prikazivanje komentara

Prikazivanje komentara se radi koristeći FirebaseRecyclerAdapter koji je prethodno objašnjen u poglavlju 3.3.2. Dodatna funkcionalnost koja će se ovdje objasniti je ViewSwitcher okvir sučelja (eng. Layout frame) koji omogućava prebacivanje između dva pogleda (eng. Views) unutar iste aktivnosti. Različiti pogledi se pokazuju u ovisnosti o tome postoje li komentari za određeni proizvod.

Za implementaciju funkcionalnosti je potrebno kreirati instancu klase ViewSwitcher i obaviti povezivanje sa GUI elementom odnosno okvirom sučelja. (Ispis 18)

```
ViewSwitcher viewSwitcher;
View myFirstView,mySecondView;

viewSwitcher = (ViewSwitcher)findViewById(R.id.viewSwitcherProduct);
myFirstView= findViewById(R.id.switch_product_one);
mySecondView = findViewById(R.id.switch_product_two);
```

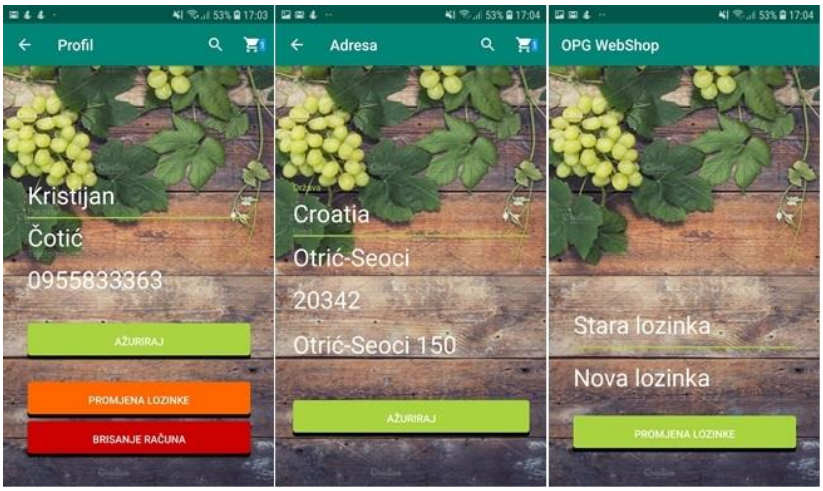
Ispis 18: Povezivanje ViewSwitcher okvira sučelja

Promjena sa jednog na drugi pogled se radi pomoću metoda showPrevious() i showNext() koje pripadaju klasi ViewSwitcher u ovisnosti o tome da li komentari postoje ili ne.

3.5. Informacije o korisniku i adresi

Aktivnost korisničkog profila služi za pregled i unos informacija o korisniku poput imena, prezimena i broja mobitela i mogućnosti promjene lozinke i brisanja računa. Ako je korisnik već unio svoje podatke u poljima za unos ispisuju mu se prethodno uneseni podaci, a u suprotnom mu se pojavljuju prazna polja što je isti slučaj i kod aktivnosti za upravljanje adresom.

Ako korisnik želi izbrisati korisnički račun pojavljuje mu se skočni prozor koji ga pita da li je siguran da želi izbrisati račun. Klikom na botun za promjenu lozinke otvara se nova aktivnost unutar koje se od korisnika traži unos stare i novu lozinku. Na slici Slika 8 prikazane su aktivnosti adrese, profila i promjene lozinke.



Slika 8: Aktivnosti profila, promjene lozinke i adrese

3.5.1. Funkcionalnosti dodavanja i ažuriranja profila i adrese

Za spremane podataka koje je korisnik popunio unutar EditText polja u bazu, potrebno je kreirati referencu na mjesto u bazi gdje se žele spremati podaci i definirati podatke koji se žele spremiti ili izmijenit.

HashMap objekt se koristi za dodavanje podataka koji se žele izmijeniti. Podaci se dodaju u obliku ključ-vrijednost. Izmjena unutar baze se radi pomoću metode updateChildren() klase DatabaseReference kojoj se kao parametar proslijedi HashMap objekt kao što je prikazano u ispisu Ispis 19. Korisnički podaci se u bazu spremaju pod jedinstvenim ključem korisnika pomoću getUid metode koja pripada FirebaseAuth klasi. Na taj način se povezuje određeni korisnik s njegovim podacima.

```
Map<String, Object> user_addr_update = new HashMap<>();
user_addr_update
.put(Auth_user.getUid()+"/state",state.getText().toString() );
user_addr_update
.put(Auth_user.getUid()+"/city",city.getText().toString() );
user_adrs.updateChildren(user_addr_update);
```

Ispis 19: Dodavanje ili ažuriranje podataka u bazu

3.5.2. Promjena lozinke

Promjena lozinke se izvršava na način da se dohvate korisnikove vjerodajnice koristeći adresu i staru lozinku njegove elektroničke pošte. Nakon toga je korisnika potrebno ponovno autentificirati koristeći metodu reauthenticate klase FirebaseUser. (Ispis 20).

```
AuthCredential credential = EmailAuthProvider.getCredential
(email, old_password.getText().toString());
mdialog.show();
user.reauthenticate(credential).addOnCompleteListener(new
OnCompleteListener<Void>() {
    });
```

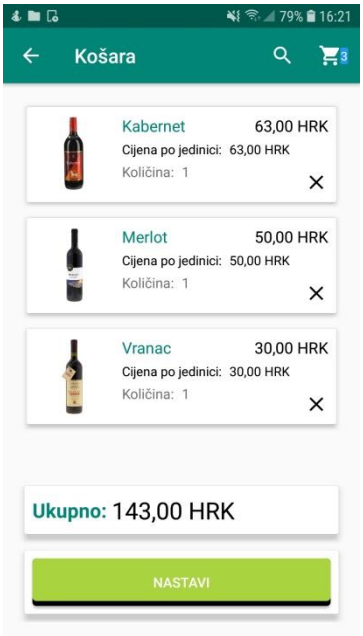
Ispis 20: Dohvaćanje vjerodajnica

Ako je ponovna autentifikacija uspješno izvršena definira se nova lozinka koja se proslijedi updatePassword()metodi čime je završen proces promjene lozinke.

3.6. Košarica

Pristup košarici je omogućen iz svake aktivnosti dodirom na botun koji se nalazi u akcijskoj traci. Aktivnost sadrži listu odabranih proizvoda u kojoj se nalazi popis proizvoda zajedno s osnovnim informacijama o svakom proizvodu. Korisnik ima mogućnost uklanjanja proizvoda iz košarice pritiskom na X strelicu smještenu na vrhu desne strane svake stavke unutar liste.

Na dnu zaslona se nalazi prikaz ukupne cijene proizvoda koja se ažurira u ovisnosti o tome da li je proizvod uklonjen iz košarice i botun koji nas vodi na Checkout aktivnost. Korisničko sučelje je prikazano na slici Slika 9.



Slika 9: Sučelje košarice

3.6.1. Dohvaćanje i prikaz proizvoda

Podaci se dohvaćaju iz lokalne baze čije je kreiranje objašnjeno u poglavlju 3.4.2. Za dohvaćanje podataka potrebno je kreirati listu Order objekta unutar koje se spremaju dohvaćeni podaci. Sam proces dohvaćanja prikazan je u ispisu Ispis 21 koji se sastoji od kreiranja instance baze i korištena njene metode getCart().

```
List<Order> carts= new ArrayList<>();
carts = new Database(this).getCarts();
```

Ispis 21: Dohvaćanje podataka iz baze

Proizvodi u košarici se prikazuju koristeći RecyclerView za koji je potrebno proširiti ViewHolder klasu kojom je predstavljen svaki pogled unutar RecyclerView-a. Tim pogledima upravlja Adapter klasa koja koristeći metodu onCreateViewHolder() kreira poglede po potrebi. Metoda onBindViewHolder() dohvaća prikladne podatke iz liste i s njima popunjava svaki od ViewHolder-a.

3.6.2. Brisanje iz košarice

Brisanje iz košarice se radi unutar onBindViewHolder() metode koja pripada klasi RecycerViewAdater pozivanjem metode removeItem() koja iz liste objekta ukloni određeni element u ovisnosti o poziciji unutar RecyclerView-a što možemo vidjeti na ispisu Ispis 22.

```
public void removeItem(int position)
{
    listData.remove(position);
    notifyItemRemoved(position);
}
```

Ispis 22: Metoda za uklanjanje iz košarice

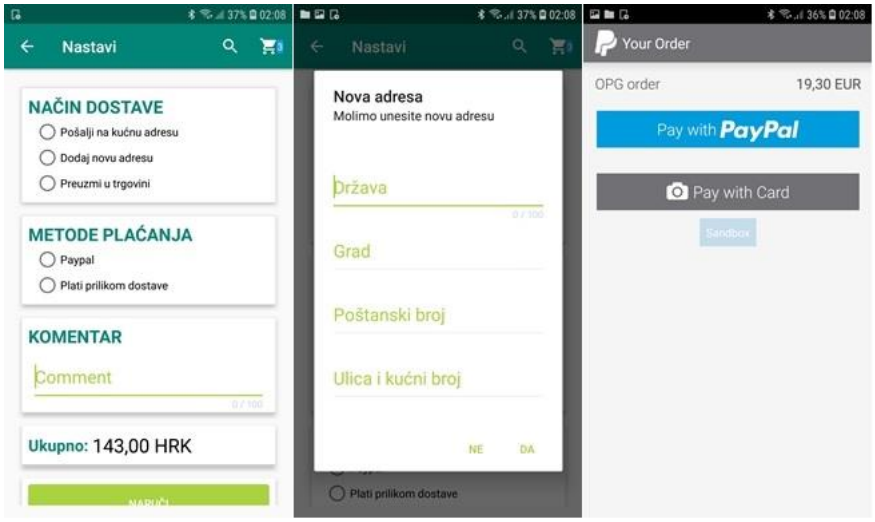
3.7. Proces naručivanja

Klikom na botun „Nastavi“ unutar košarice prelazimo na aktivnost provjere narudžbe. Unutar ove aktivnosti korisnik može odabrati način dostave i plaćanja narudžbe te ostaviti komentar prodavaču.

Korisnik koristeći radio botune može odabrati dostavu na kućnu adresu, specificirati neku drugu adresu ili odabrati preuzimanje narudžbe u trgovini. Ako kupac

odabere dostavu na drugu adresu pojavi se skočni prozor unutar kojeg se nalaze polja za unos adrese i botun za potvrdu. Prilikom odabira dostave ako cijena proizvoda u košari ne prelazi iznos od 500 kuna, korisnik ne plaća dostavu. U suprotnom iznos dostave iznosi 30 kuna i korisniku se ispisuje poruka o plaćanju dostave. U slučaju odabira preuzimanja u trgovini korisnik ne plaća dostavu.

Kod odabira načina plaćanja postoje dva radio botuna za plaćanje prilikom dostave i koristeći Paypal. Ako korisnik odabere Paypal plaćanje u završnom procesu pritiskom na botun naruči otvara se nova aktivnost (Paypal) unutar koje korisnik unosi podatke o Paypal računu i potvrđuje plaćanje. Ukoliko je plaćanje uspješno izvršeno ispiše mu se poruka da je narudžba uspješno poslana. Sučelje Checkout aktivnosti je prikazano na slici Slika 10.



Slika 10: Checkout aktivnost

3.7.1. Postupak kreiranja narudžbe

Kako bi se kreirala narudžba potrebno je kreirati novi model podataka odnosno klasu OrderRequest koja se koristi za spremanje podataka o detaljima narudžbe. Sam proces spremanja podataka u bazu se neće posebno pojašnjavati jer je objašnjen u poglavlju 3.4.3.

Odabir odgovarajućih podataka o dostavi i plaćanju se radi pomoću radio botuna koji se kreiraju instanciranjem objekta klase RadioButton i njihovim povezivanjem sa GUI elementima.

setOnCheckedChangeListener metoda je zadužena za definiranje akcija koje će se dogoditi u slučaju odabira određenog radio botuna. Primjerice u ovisnosti o odabiru određenog botuna mogu se postaviti vrijednost varijabli za način dostave, plaćanja i ukupne cijene (Ispis 23).

```
radioPickInStore.setOnCheckedChangeListener(new
CompoundButton.OnCheckedChangeListener() {
    @Override
    public void onCheckedChanged
(CompoundButton buttonView, boolean isChecked) {
        if(isChecked)
        {
            shipping=0;
            shippment.setVisibility(View.GONE);
delivery_opt=getResources().getString(R.string.pick_up_in_store);
        }
    }
});
```

Ispis 23: Postavljanje varijabli za način dostave

3.7.2. Paypal plaćanje

Za korištenje Paypal plaćanja se koristi PayPal-Android-SDK [5] biblioteka. Prije same implementacije potrebno je na Paypal Developer stranici kreirati računu kako bi se dobio API ključ. Prilikom testiranja koristimo testne račune za primanje uplata i plaćanje.

PaypalConfiguration klasom prikazanom u ispisu Ispis 24 izvršava se konfiguracija samog Api-a postavljanjem API ključa koji identificira određenu aplikaciju. Također omogućava definiranje okruženja u kojem se Paypal koristi, a može se odnositi na testno ili produkcijsko okruženje.

```
PayPalConfiguration payPalConfiguration = new
PayPalConfiguration().environment(ENVIRONMENT_SANDBOX).
clientId("AYegTkx5X8CQCpAb2AhvVCHLFEADS0ixPwVGrcIV7QSeorpF9tC2M5");
```

Ispis 24: Paypal konfiguracija

Postupak plaćanja prikazan u ispisu Ispis 25 se sastoji od postavljanja PayPalPayment objekta koji sadrži informacije o cijeni, valuti plaćanja i naslovu narudžbe. Pomoću njega se te informacije prosljeđuju u Paypal aktivnost čime se izvršava samo plaćanje.

```
PayPalPayment payPalPayment=new PayPalPayment(new BigDecimal(converted)
,"EUR","OPG order",PayPalPayment.PAYMENT_INTENT_SALE);
Intent intent=
new Intent(getApplicationContext(),PaymentActivity.class);
intent.putExtra(PayPalService.EXTRA_PAYPAL_CONFIGURATION,
payPalConfiguration);
intent.putExtra(PaymentActivity.EXTRA_PAYMENT,payPalPayment);
startActivityForResult(intent,PAYPAL_REQUEST_CODE);
```

Ispis 25: Paypal postupak plaćanja

Ovdje je još važno naglasiti kako Paypal ne dozvoljava plaćanje u hrvatskoj valuti koja se koristi unutar aplikacije te je stoga potrebno obaviti konverziju. Paypal podržava plaćanje u dolarima i eurima no kako se unutar aplikacije obavlja plaćanje u kunama potrebno je obaviti konverziju iz eura ili dolara u kune. Konverzija se radi koristeći API hrvatske narodne banke kako bi se dobio tečaj za određenu valutu na taj dan.

Pomoću klase StringRequest se može uhvatiti odgovor sa poslužitelja u obliku stringa koji se sastoji od tečaja određenog broja valuta. Taj string je moguće daljnjim postupkom pretvoriti u objekte koji sadrže informacije o pojedinoj valuti. Informacije iz tih objekata odnosno informacije o srednjem tečaju se koriste za pretvorbu iz jedne valute u drugu što je prikazano u ispisu Ispis 26.

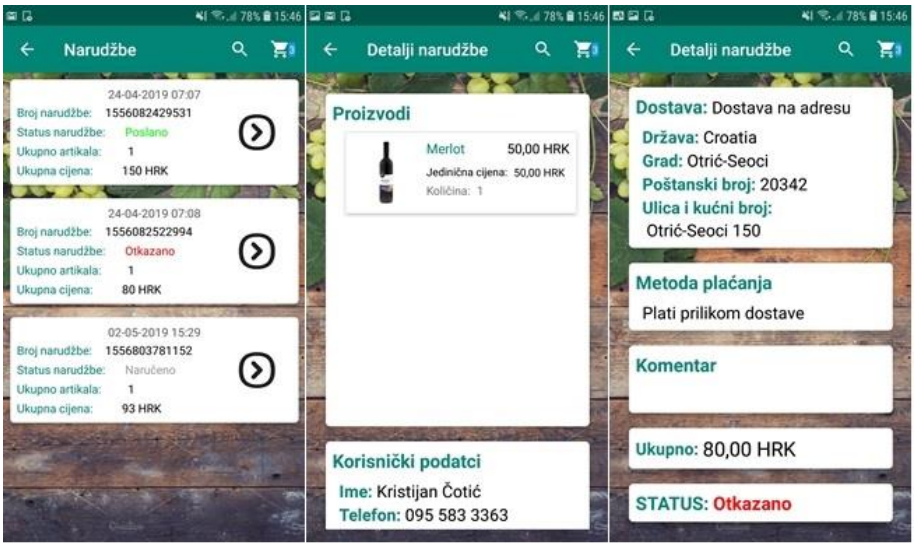
```
JSONArray jarray = new JSONArray(jsonString);
double total_return;

for (int p = 0; p < jarray.length(); p++) {

    JSONObject jsonObject = jarray.getJSONObject(p);
    Currency cur = gson.fromJson(String.valueOf(jsonObject),
Currency.class    currency.add(cur);
    if (cur.getCurrency_code().equals("EUR")) {
        total_return = total_paypal /
Double.parseDouble(cur.getMedian_rate());
            converted=String.valueOf(total_return);
    }
}
```

Ispis 26: Pretvaranje JSON stringa u objekte i konverzija cijene

3.8. Pregled povijesti i detalja narudžbe

Povijesti narudžbi se pristupa sa početnog zaslona klikom na „Moje narudžbe“ unutar bočnog izbornika. Korisnici mogu vidjeti sve svoje narudžbe sortirane po datumu. Za svaku narudžbu se prikazuju informacije o datumu, broju, statusu, ukupnom broju artikala i ukupnoj cijeni narudžbe.

Klikom na svaku pojedinu narudžbu korisniku se otvara nova aktivnost sa detaljima svake pojedine narudžbe gdje može pregledat artikle koje naručio, informacije o adresi dostave, metodi dostave i plaćanja, statusu svoje narudžbe, ostavljenom komentaru i ukupnoj cijeni. Pregled korisničkog sučelja za ove aktivnosti je prikazan na slici Slika 11.



Slika 11: Pregled narudžbi i detalji

Funkcionalnosti koje se ovdje koriste, kao što su čitanje podataka iz baze te prikaz popisa narudžbi unutar RecyclerView-a, su opisane u poglavlju 3.3.1 pa ih se neće posebno opisivati. Funkcionalnost koju je potrebno istaknuti je pretvorba numeričkog statusa narudžbe u tekst.

Kod čitanja podataka o narudžbi iz baze informacije o statusu narudžbe se pomoću if uvjeta pretvaraju u tekst i postavlja boja teksta za prikaz na zaslonu ovisno o tome da li je narudžba poslana, zaprimljena ili isporučena što se vidi iz ispisa Ispis 27.

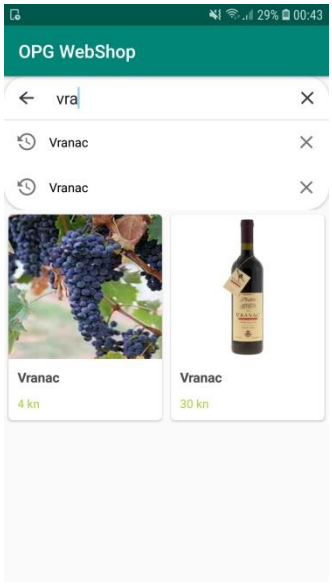
```
if(ord.getStatus().equals("0"))
{
    viewHolder.order_status.setText(R.string.order_sent_small);
    viewHolder.order_status.setTextColor(Color.parseColor("#999999"));
}
```

Ispis 27: Pretvorba numeričkog statusa u tekstualni

3.9. Pretraživanje proizvoda

Aktivnost pretraživanja proizvoda se sastoji od tražilice unutar koje korisnik pretražuje određeni proizvod. Kako korisnik upisuje naziv određenog proizvoda tako mu se

pokazuju prijedlozi. Klikom na određeni prijedlog korisniku se na ekranu prikazuju proizvodi koji odgovaraju njegovom unosu. Sučelje tražilice je prikazano na slici Slika 12.



Slika 12: Sučelje za pretraživanje

Mana Firebase usluge za pohranu podataka je pretraživanje. Pretraživanje je moguće samo ako se unese cijeli pojam što za krajnjeg korisnika i ne predstavlja neku funkcionalnost. Rješenje je koristiti MaterialSearchBar biblioteku koja osim samog grafičkog elementa za pretraživanje nudi funkcionalnosti koje su potrebne kako bi se korisniku olakšalo pretraživanje.

Za korištenje MaterialSearchBar-a potrebno je kreirati objekt odgovarajuće klase i povezati ga s njegovom reprezentacijom na GUI sučelju.

Svi proizvodi iz baze odnosno imena proizvoda se učitavaju u listu stringova na osnovu koje se korisniku prikazuju prijedlozi. Za prikazivanje prijedloga potrebno je pregaziti ponašanje onTextChanged metode koja se izvršava svaki put kada korisnik doda ili ukloni slovo iz tražilice.

Kao što je vidljivo iz ispisa Ispis 28 unutar onTextChanged metode se kreira nova lista stringova. Ta lista je bitna jer kako korisnik mijenja svoj unos ona se svaki put iznova kreira, a popunjava se na način da se korisnikov unos uspoređuje sa listom imena svih proizvoda. Ako je korisnikov unos sadržan u imenu jednog ili više proizvoda oni se prebacuju u novu listu koja se pomoću metode setLastSuggestions prikazuje korisniku.

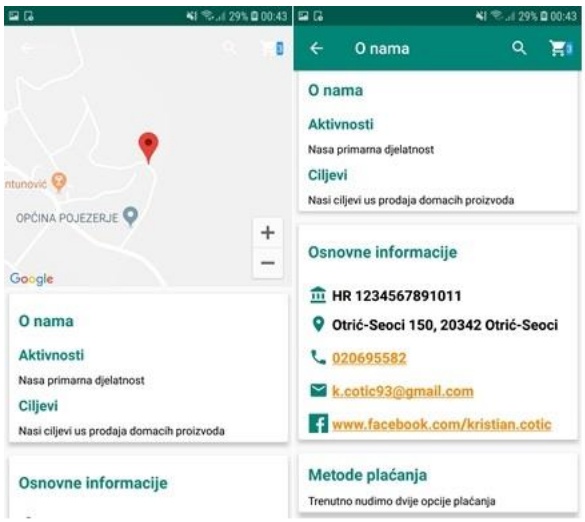
```
public void onTextChanged(CharSequence s, int start,
int before, int count) {
    final List<String>suggest=new ArrayList<>();
    for (String search:Suggestions)
    {
if(search.toLowerCase().contains(materialSearchBar.getText()
.toLowerCase()))
        {
            suggest.add(search);
            materialSearchBar.setLastSuggestions(suggest);
        }
```

Ispis 28: Prikazivanje prijedloga

3.10. Prikaz informacija o OPG-u

Unutar ove aktivnosti korisniku se prikazuju osnovne informacije o poduzeću poput lokacije, primarne djelatnosti, načinu plaćanja i kontakt informacija.

Unutar padajuće akcijske trake korisnik može vidjeti lokaciju OPG-a pomoću Google karata. Klikom na tu lokaciju otvaraju mu se aplikacija Google karte unutar koje može pokrenuti navigaciju. Klikom na kontakt informacije korisniku se otvara određena aplikacija za kontaktiranje ovisno o tome koju je kontakt informaciju kliknuo. Sučelje ove aktivnosti je prikazano na slici Slika 13.



Slika 13: Prikaz zaslona O nama

3.10.1. Prikaz Google karata

Za prikaz karata unutar padajuće akcijske trake unutar XML datoteke je potrebno kreirati dva okvira sučelja AppBarLayout i CollapsingToolbarLayout koji je dizajniran kao dijete prvoga i unutar njega smjestiti element za prikaz mape prikazan u ispis Ispis 29.

```
<com.google.android.gms.maps.MapView
    android:id="@+id/main_activity_mapview"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:clickable="true"
    app:layout_collapseMode="parallax" />
```

Ispis 29: GUI element za prikaz mape

Zatim je potrebno kreirati MapView objekt i koristeći njegovu metodu getMapAsync pokreće se prikaz karte.

Ponašanje karata odnosno prikaz zadanih koordinata i oznake na karti se može prilagoditi prema vlastitim zahtjevima na način da se pregazi ponašanje OnMapReady metode prikazane u ispisu Ispis 30.

```
public void onMapReady(GoogleMap googleMap) {

        LatLng position = new LatLng(43.157406, 17.459833);
        Marker marker = googleMap.addMarker(new
MarkerOptions().position(position).title(markerText));

    CameraUpdate cameraUpdate = CameraUpdateFactory.newLatLngZoom
(position, 16);
    googleMap.animateCamera(cameraUpdate);
    googleMap.setPadding(0,0,0,50);
}
```

Ispis 30: OnMapReady metoda

3.10.2. Pokretanje vanjskih aplikacija

Kako bi se omogućilo pokretanje aplikacija poput telefona i elektroničke pošte i prosljeđivanje informacija iz OPG aplikacije tim istim aplikacijama TextView elementu na koji se ispisuju te informacije potrebno je dodati autoLink atribut i postaviti mu određenu vrijednost ovisno o tome koja se vanjska aplikacija želi pokrenuti.

4. Korisničko sučelje funkcionalnosti aplikacije za prodavača

U ovom će se opisati izrada aplikacije namijenjene prodavačima. Prikazat će se izgled korisničkog sučelja i funkcionalnosti aplikacije. Kako je dosta funkcionalnosti isto kao i u slučaju aplikacije za kupce te funkcionalnosti se neće dodatno pojašnjavati. Pojasnit će se samo funkcionalnosti koje su specifične za ovu aplikaciju.

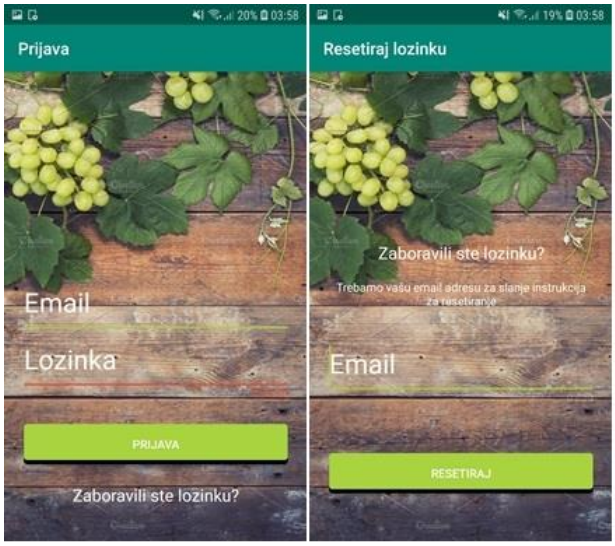
4.1. Prijava korisnika i resetiranje lozinke

Prijava korisnika se sastoji od tekstualnih polja za unos adrese i lozinke elektroničke pošte, botuna za prijavu i tekstualne poveznice na aktivnost za resetiranje lozinke. Kod ove aplikacije se ne koristi prijava putem Google autentifikacije.

Ako korisnik ne popuni sva polja ili unese krivu adresu ili lozinku o tome ga se obavještava porukom na zaslonu, a u suprotnom mu se dozvoljava prijava i prelazak na aktivnost početnog izbornika.

Ako korisnik zaboravi lozinku, klikom na tekstualnu poveznicu mu se otvara aktivnost unutar koje treba unijeti adresu svoje elektroničke pošte, na koju će mu se poslati link za resetiranje lozinke. Sučelje prijave i resetiranja lozinke je prikazao na slici Slika 14.

Implementacija funkcionalnosti koja se ovdje koristi je prikazana u poglavlju 3.2.2

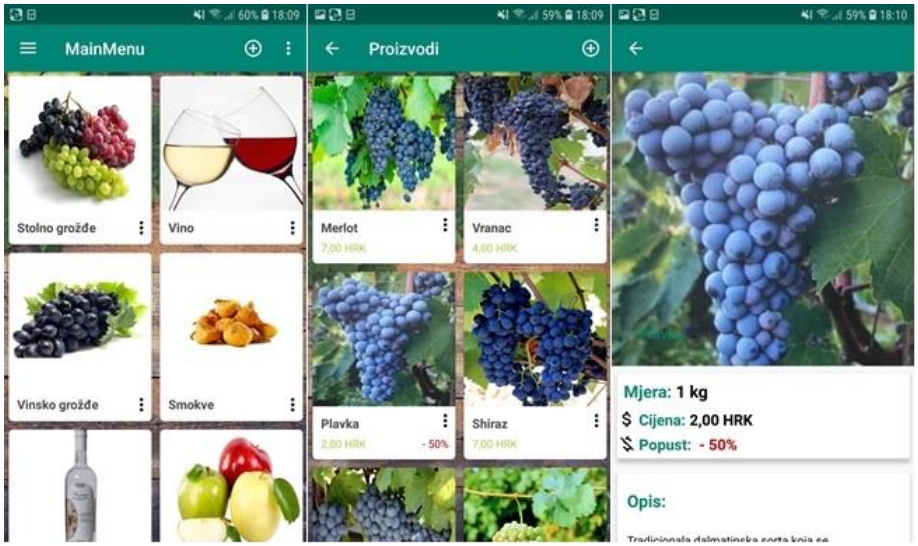


Slika 14: Prijava i resetiranje lozinke

4.2. Kategorije, proizvodi i detalji proizvoda

Kada se korisnik uspješno prijavi otvara mu se glavni izbornik unutar kojeg su prikazane kategorije i akcijska traka u kojoj se nalaze botuni za dodavanje proizvoda, otvaranje bočnog izbornika i botun za kreiranje računa za prodavača. Kreiranje novog računa je moguće jedino u slučaju ako je prijavljeni korisnik administrator.

Za svaku kategoriju uz ime postoji botun u obliku 3 točkice. Klikom na njega korisniku se otvara skočni prozor unutar kojeg može odabrati opcija za osvježavanje ili brisanje odabrane kategorije. Svaka kategorija koju korisnik može odabrati nudi prethodno spomenute mogućnosti. Klikom na proizvod otvara se aktivnost sa detaljima proizvoda gdje je moguće komentiranje i ažuriranje pojedinog proizvoda. Izgled ovih aktivnosti je prikazani su na slici Slika 15.



Slika 15: Sučelje kategorija, proizvoda i detalja

4.2.1. Implementacija skočnog prozora

Kako bi se pokrenuo skočni prozor unutar xml datoteke koja predstavlja svaku kategoriju, potrebno je dodati ImageView element i pomoću setOnClickListener metode povezati ga sa određenim ponašanjem.

To ponašanje odnosno pokretanje skočnog izbornika implementirano je unutar showMenu metode, koja kao parametar mora prihvatiti ključ određene kategorije kako bi se znalo na koju se kategoriju kliknulo.

Ispis 31 prikazuje implementaciju skočnog prozora za kojeg je potrebno kreirati objekt klase PopupMenu i njegovoj inflate() metodi proslijediti xml sučelje izbornika. Pokretanje određenih radnji u ovisnosti na koju je stavku unutar izbornika korisnik kliknuo se radi unutar setOnMenuItemClickListener metode.

```
public void showMenu(View view, final String pos) {
    PopupMenu popup = new PopupMenu(this,view);
    popup.inflate(R.menu.menu_click);
    popup.setOnMenuItemClickListener(new
PopupMenu.OnMenuItemClickListener() {
        @Override
        public boolean onMenuItemClick(MenuItem item) {
            switch (item.getItemId()) {
                case R.id.action_update:

                    return true;
                case R.id.action_delete:

                    return true;
                default:
            }
            return false;
        }
    });
```

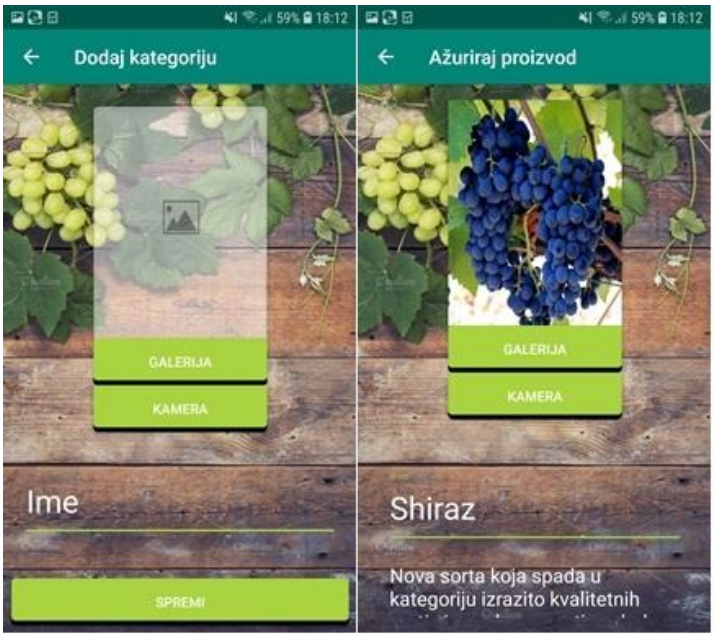
Ispis 31: Metoda za pokretanje skočnog izbornika

4.3. Dodavanje i osvježavanje kategorija i proizvoda

Korisničko sučelje ovih aktivnosti se sastoji od prikaza za sliku, botuna za odabir slike koristeći kameru ili galeriju i polja za unos informacija o kategorijama ili proizvodima.

Klikom na botun Kamera korisniku se otvara aplikacija kamera unutar koje može slikati i ta mu se slika prikazuje na zaslonu. Također pritiskom na botun galerija, korisnik može odabrati sliku iz galerije. Korisnik zatim popunjava polja za unos informacija o kategoriji ili proizvodi i pritiskom na botun „Spremi“ sprema podatke u bazu.

Izgled aktivnosti za dodavanje i ažuriranje kategorija i proizvoda prikazan na slici Slika 16 je identičan i jedina razlika je da su kod ažuriranja polja unaprijed popunjena sa podacima iz baze.



Slika 16: Izgled sučelja za dodavanje i ažuriranje proizvoda

4.3.1. Dohvaćanje slike iz galerije

Za pokretanje galerije i dohvaćanje određene slike potrebno je kreirati Intent objekt. Koristeći njegove metode potrebno je postaviti o kojem tipu Intenta se radi i akciju koja se u ovom slučaju odnosi na dohvaćanje sadržaja. Intent objekt se proslijedi metodi startActivityForResult kako bi se pokrenula aktivnost galerije kao što je prikazano na ispisu Ispis 32.

```
Intent intent = new Intent();
intent.setType("image/*");
intent.setAction(Intent.ACTION_GET_CONTENT);
startActivityForResult(Intent.createChooser(intent, "Select Picture"),
PICK_IMAGE);
```

Ispis 32: Pokretanje galerije

Rezultat, odnosno lokacija slike, se sprema u odgovarajući objekt na način da se pregazi ponašanje onActivityResult metode. Ta metoda se poziva svaki put kada je potrebno uhvatiti rezultat koji neka aktivnost vrati. Unutar te aktivnosti je također implementirano spremanje slike u Firebase Storage, usluzi čija ja je implementacija prikazana u poglavlju 4.3.3.

4.3.2. Dohvaćanje slike koristeći kameru

Razlika između dohvaćanja slike koristeći galeriju i kameru je u tome što se za sliku slikanu kamerom mora kreirati privremena datoteka odnosno fizička lokacija na koju će se slika spremiti.

Postupak kreiranja privremene datoteke prikazan na ispisu Ispis 33. Definira se direktorij unutar kojeg se slika sprema i kreira se jedinstveni naziv za svaku sliku.

```
String timeStamp = new SimpleDateFormat("yyyyMMddHHmmss")
.format(new Date());
String pictureFile = "ZOFTINO_" + timeStamp;
File storageDir =
context.getExternalFilesDir(Environment.DIRECTORY_PICTURES);
File image = File.createTempFile(pictureFile, ".jpg", storageDir);
```

Ispis 33: Kreiranje privremene datoteke

Pokretanje kamere prikazano u ispisu Ispis 34 se sastoji od toga da se kreira Intent objekt i koristeći njegovu putExtra metodu u aktivnost kamere se proslijedi privremena datoteka koja je prethodno kreirana. Tu privremenu datoteku je prethodno potrebno pretvoriti u Uri objekt. Daljnji postupak hvatanja rezultata aktivnosti kamere je isti kao i kod dohvaćanja slike iz galerije.

```
Intent cameraIntent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
Uri photoURI = FileProvider.getUriForFile(context,
        "com.example.kristijan.opgwebshopadmin.fileprovider",
        pictureFile);
cameraIntent.putExtra(MediaStore.EXTRA_OUTPUT, photoURI);
startActivityForResult(cameraIntent, cameraIntent.
getRequestCaptureImage());
```

Ispis 34: Pokretanje kamere

4.3.3. Spremanje slike u Firebase Storage

Kako se u Firebase uslugu pohrane podataka ne može spremiti slika, već veza na određenu sliku u obliku stringa, potrebno je koristiti Firebase Storage koji omogućava spremanje slike na poslužitelj i dohvaćanje veze na tu sliku.

Za korištenje Firebase Storage-a potrebno je kreirati instancu klase FirebaseStorage i pomoću klase StorageReference odrediti lokaciju na koju će se spremiti slika (Ispis 35).

```
Firestorage storage = FirebaseStorage.getInstance();
String image_name = UUID.randomUUID().toString();
final StorageReference imageFolder = storageReference
.child("images/" + image_name);
```

Ispis 35: Korištenje Firebase Storage

Ispis 36 pokazuje dodatni korak odnosno kompresiju slike spremljene u Bitmap formatu prije spremanja u Firebase storage. Slika se kompresira jer nije potrebna velika kvaliteta slike a samim time se skraćuje i vrijeme potrebno da se slika spremi na poslužitelj.

```
ByteArrayOutputStream baos = new ByteArrayOutputStream();
bmp.compress(Bitmap.CompressFormat.JPEG, 25, baos);
byte[] data = baos.toByteArray();
```

Ispis 36: Kompresija slike

Pomoću putBytes metode slika se sprema u Firebase storage. Kako bi se pratilo da li je slika uspješno spremljena ili ne koristi se addOnSuccessListener metoda. U slučaju uspješnog spremanja slike pomoću metode getDownloadUrl se dohvaća veza na sliku (Ispis 37).

```
imageFolder.putBytes(data).addOnSuccessListener(new
OnSuccessListener<UploadTask.TaskSnapshot>() {
    @Override
    public void onSuccess(UploadTask.TaskSnapshot taskSnapshot) {
          imageFolder.getDownloadUrl()
          .addOnSuccessListener(new OnSuccessListener<Uri>() {
            @Override
            public void onSuccess(Uri uri) {
                link = uri.toString();
            }
        });
    }
})
```

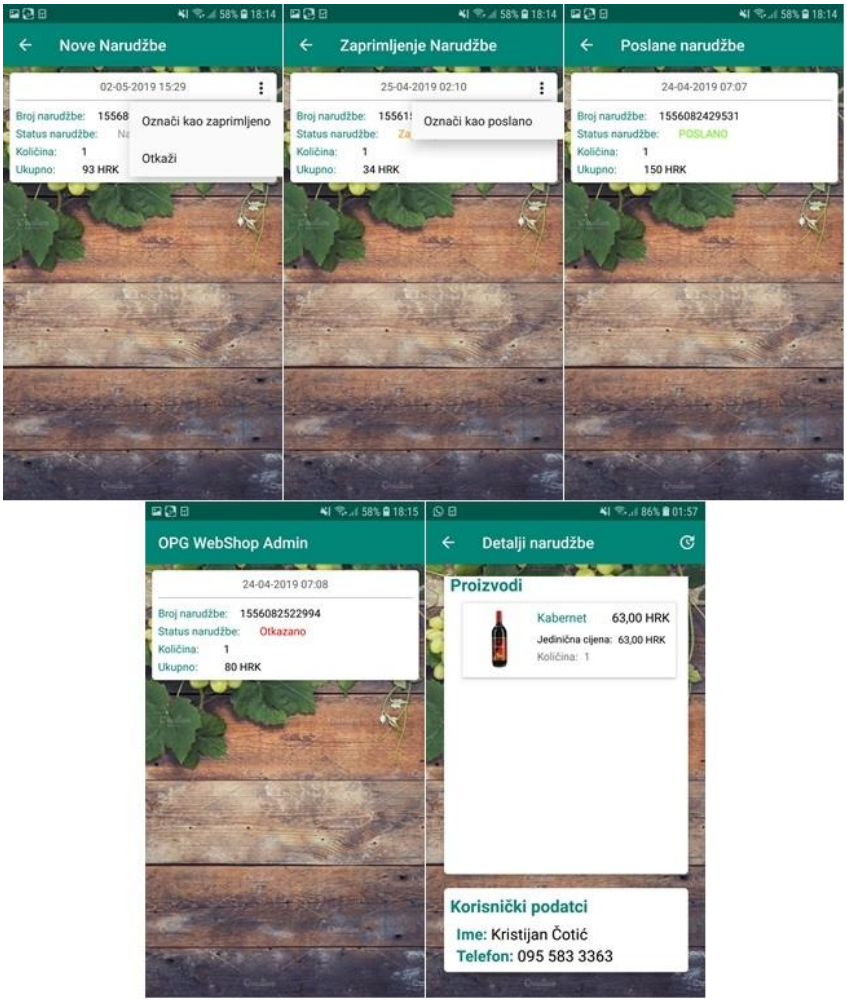
Ispis 37: Spremanje slike i dohvaćanje veze na sliku

4.4. Upravljanje narudžbama

Za upravljanje narudžbama se koriste 4 aktivnosti. Razlika između te 4 aktivnosti je u tome što prikazuju različite statuse narudžbi odnosno nove, zaprimljene, poslane i otkazane narudžbe.

Aktivnosti se sastoje od prikaza liste narudžbi i pritiskom na određenu narudžbu korisnik pristupa novoj aktivnosti koja prikazuje detalje pojedine narudžbe. U ovoj aktivnosti prodavač može vidjeti koje artikle je kupac naručio i koju je metodu dostave i plaćanja izabrao. Ova aktivnost je ista kao u poglavlju 3.8.

Kod aktivnosti novih narudžbi korisniku se klikom na tri točkice, koje se nalaze na prikazu svake narudžbe, otvara skočni izbornik gdje ima mogućnost promjene statusa narudžbe. Status se može promijeniti u zaprimljeno ili otkazano. Npr. promjenom statusa narudžbe iz nove u zaprimljenu ta narudžba se uklanja iz aktivnosti novih narudžbi i prelazi u aktivnost zaprimljenih. Također kod zaprimljenih narudžbi status je moguće promijenit u poslano. Na slici Slika 17 prikazano je sučelje ovih aktivnosti.



Slika 17: Sučelje upravljanja narudžbama

Promjena statusa narudžbe se radi unutar populateViewHolder metode opisane u poglavlju 3.3.1 koristeći getItem metodu koja dohvati određenu narudžbu i spremi je u

odgovarajući objekt. Taj objekt, i ključ pod kojim je spremljen, u bazi se proslijedi showMenu metodi opisanoj u poglavlju 4.2.1. Unutar koje se radi promjena vrijednosti statusa i ponovno spremanje u bazu, kao što je prikzano na primjeru Ispis 38.

```
current.setStatus("2");
order_request.child(pos).setValue(current);
```

Ispis 38: Promjena statusa i spremanje u bazu

Dvije bitne funkcionalnosti koje će se ovdje opisati su kreiranje PDF računa i slanje toga računa elektroničkom poštom, što se radi kod promjene statusa narudžbe u zaprimljeno.

4.4.1. Kreiranje PDF računa

Kako bi se kreirao PDF koristi se iText G [6] biblioteka koja je zaista ogromna biblioteka za rad sa PDF formatom datoteka. Dio te biblioteke koji se koristi je popunjavanje unaprijed kreirane PDF forme.

Prije samog početka implementacije potrebno je kreirati PDF formu koristeći Adobe Acrobat pro. Formu se kreira tako da se dodaju i slažu polja kojima se dodjeli jedinstveni ID. Tako kreiranu formu je potrebno smjestiti u raw mapu koja je kreirana unutar res mape.

Za popunjavanje PDF-a je kreirana PDF klasa i konstruktor koji kao parametre prima OrderRequest objekt i poziciju narudžbe. Popunjavanje forme se radi unutar createpdf metode.

Postupak popunjavanja forme se sastoji od toga da se kreira instanca klase PdfReader koja čita formu i privremena datoteka (Poglavlje 4.3.2) u koju će se spremati popunjena forma. PdfStamper klasa i njene metode se koristi za dohvaćanje polja koja se nalaze u formi i spremanje popunjene forme u privremenu datoteku. Ispis 39 prikazuje postupak dohvaćanja polja forme.

```
PdfReader reader = new PdfReader(context.getResources()
.openRawResource(R.raw.opgtemplate4));

pdfFile=createImageFile(context);
PdfStamper stamper = new PdfStamper(reader,
        new FileOutputStream(pdfFile));
AcroFields form = stamper.getAcroFields();
```

Ispis 39: Dohvaćanje polja forme

Kada se dohvate polja ona se popunjavaju pomoću setField metode koja pripada AcroFields klasi tako da joj se kao parametri proslijede naziv polja u formi i vrijednost za pojedino polje kao što je pokazano u ispisu Ispis 40.

```
form.setField("address", orderRequest.getUser().getStreetHouseNum());
form.setField("city", orderRequest.getUser().getPostalCode()+"
"+orderRequest.getUser().getCity());
```

Ispis 40: Spremanje vrijednosti u polja

Kreirani PDF odnosno račun se sprema u Firebase Storage kako bi se veza na taj račun mogla poslati korisniku putem elektroničke pošte.

4.4.2. Slanje PDF-a elektroničkom poštom

Da bi se automatizirao proces slanja elektroničke pošte koristi se JavaMail API. Kako bi se JavaMail API koristio potrebno je unutar lib mape projekta smjestiti tri jar datoteke: activation.jar, additionnal.jar i mail.jar. Također je potrebno implementirati GmailSender klasu koja omogućava slanje elektroničke pošte koristeći Gmail smtp server.

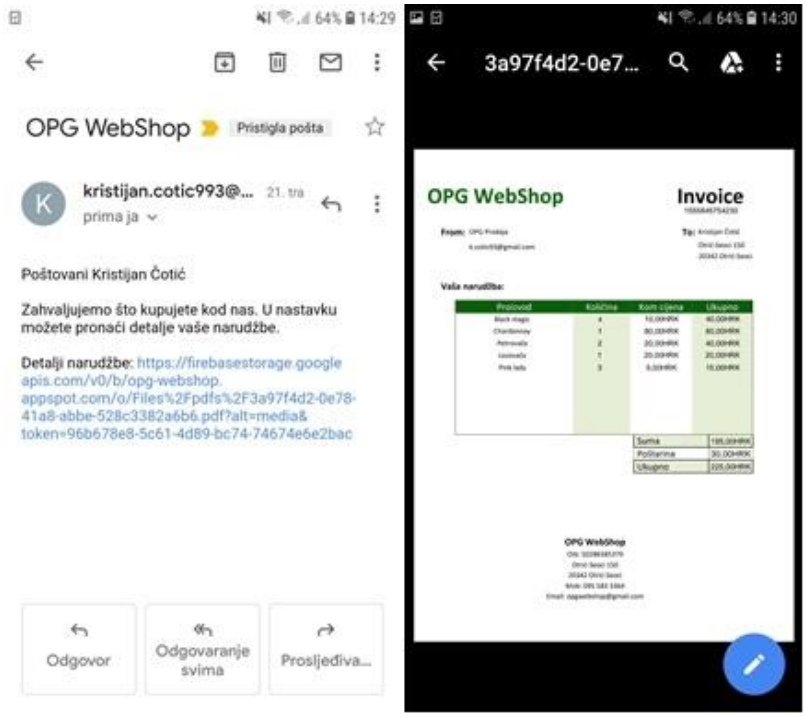
Kôd unutar ove klase se odvija u novoj niti (eng. *thread*) kako se tijekom procesa slanja elektroničke pošte ne bi blokirala glavna nit.

GMailSender je klasa koja se koristi za prijavu na račun s kojega se šalje elektroničkom poštom. Prijava se radi pomoću lozinke u ne kodiranom obliku što i nije dobra praksa, ali za potrebe ove aplikacije kodiranje nije nužno implementirati. Pomoću njene sendEmail metode se izvršava slanje elektroničke pošte tako da joj se proslijedi naslov i tijelo elektroničke pošte, adresu pošiljatelja i primatelja elektroničke pošte. U ispisu Ispis 41 je prikazano kreiranje i slanje elektroničke pošte.

```
String body="Poštovani"+" "+order.getUser().getName()+"
"+order.getUser().getSurname()+"\n\nZahvaljujemo što kupujete kod nas.
 U nastavku možete pronaći detalje vaše narudžbe.\n\nDetalji narudžbe:
"+link;
GMailSender sender = new GMailSender
        ("kristijan.cotic993@gmail.com","torcida19500");
sender.sendMail("OPG WebShop", body,
        "kristijan.cotic993@gmail.com", order.getEmail());
```

Ispis 41: Kreiranje i slanje elektroničke pošte

Na slici Slika 18 je prikaza elektronička pošta koju korisnik primi i PDF koji je preuzeo klikom na link koji se nalazi unutar elektroničke pošte.



Slika 18: Primljena elektronička pošta i preuzeti PDF

4.5. Ažuriranje informacija o OPG-u

Na slici Slika 19 je prikazana aktivnost ažuriranja informacija o OPG-u koja se sastoji od polja za unos osnovnih informacija kao što su: primarne aktivnosti i ciljevi, adresa, IBAN, elektronička pošta, broj telefona i Facebook stranica.

Ako je korisnik već prethodno unio podatke o OPG-u ti podaci se dohvaćaju iz baze i prikazuju korisniku koji ih može izmijenit i ponovo spremiti, a u suprotnom mu se prikazuju prazna polja koja treba popuniti i klikom na „Spremi“ spremiti u bazu. Funkcionalnost koja se krije iza ovog procesa je opisana u poglavlju 3.5.1.

Slika 19: Ažuriranje informacija o OPG-u

5. Dodatne funkcionalnosti i struktura baze

U ovom poglavlju će se opisati dodatne funkcionalnosti kao što su: bazna aktivnost i zašto se koristi, kreiranje i slanje obavijesti koristeći Firebase Cloud Messaging Api i struktura podataka u Firebase bazi.

5.1. Bazna aktivnost

Baznu aktivnost se koristi kako bi se u nju smjestile određene klase koje će se koristiti u svim aktivnostima aplikacije. Kako bi ostale aktivnosti mogle koristiti funkcionalnosti implementirane unutar bazne klase svaka aktivnost mora naslijediti tu klasu.

U ovom slučaju unutar bazne aktivnosti je implementirana provjera dostupnosti mreže. Provjera se radi u dva koraka i najprije se provjerava je li korisnik uopće povezan na internet koristeći neku od dostupnih metoda povezivanja, i u koliko je povezan provjerava se je li pristup Internetu dostupan.

5.1.1. Provjera mreže

Za implementaciju provjere mreže kreira se CheckConnectivity instanca klase i metode isNetworkConnectionAvailable i TestInternet.

Unutar metode isNetworkConnectionAvailable se koriste klase ConnectivityManager i NetworkInfo koje omogućuju dohvaćanje i spremanje informacije o aktivnim metodama spajanja. Metoda isConnected koja će vratiti null vrijednost ako se ne koristi ni jedna metoda spajanja i u tom slučaju će se korisnika obavijestiti da uključi neku od dostupnih metoda povezivanja. Na ispisu Ispis 42 je prikazano dohvaćanje informacija o aktivnim metodama spajanja.

```
ConnectivityManager cm = (ConnectivityManager)context
.getSystemService(context.CONNECTIVITY_SERVICE);

NetworkInfo activeNetwork = cm.getActiveNetworkInfo();
boolean isConnected = activeNetwork != null &&
        activeNetwork.isConnected();
```

Ispis 42: Dohvaćanje informacije o aktivnim metodama spajanja

Za provjeru dostupnosti pristupa Internetu koristi se TestInternet metoda koja se odvija u zasebnoj niti. Proces se odvija pomoću objekta klase HttpURLConnection kojim se ostvaruje veza prema nekom Internet resursu. Ako veza nije uspješno ostvarena korisnika se obavještava o tome da pristup internetu možda nije dostupan. Provjera pristupa određenom Internet resursu je prikazana na ispisu Ispis 43.

```
URL url = new URL("http://www.google.com");
HttpURLConnection urlc = (HttpURLConnection) url.openConnection();
urlc.setConnectTimeout(3000);
urlc.connect();
if (urlc.getResponseCode() == 200) {
    return true;
}
```

Ispis 43: Provjera pristupa Internet resursu

5.2. Slanje i primanje obavijesti

Kako bi se moglo obavijestiti prodavača o novoj narudžbi ili kupca o statusu njegove narudžbe, koriste se obavijesti (eng. *push notification*). Obavijesti se šalju samo korisniku koji je naručio određeni proizvod.

Za primanje i slanje obavijesti koristi se FCM (Firebase Cloud Messaging). Za korištenje FCM u Gradle skriptu treba dodati Cloud Messaging biblioteku.

5.2.1. Primanje obavijesti

Obavijesti se primaju tako da se kreira novi servis (eng. *service*) za što je potrebno proširiti FirebaseMessagingService klasu. Bitno je pregaziti ponašanje onNewToken i onMessageReceived metoda.

onNewToken metoda se sama poziva svaki put kada se generira novi token i unutar nje se odvija spremanje token-a u bazu. Token je ključ odnosno jedinstveni identifikator koji se koristi za razlučivanje kojem korisniku se šalje obavijest. Generiranje token-a se radi prilikom prijave korisnika pomoću getInstanceId()metode.

```
FirebaseDatabase db = FirebaseDatabase.getInstance();
DatabaseReference tokens=db.getReference("Tokens");
Token token=new Token(tokenRefreshed,true);
tokens.child(FirebaseAuth.getInstance().
getCurrentUser().getUid()).setValue(token);
```

Ispis 44: Spremanje token-a u bazu

Prikaz obavijesti korisniku je implementiran unutar onMessageReceived metode koja sluša i hvata nadolazeće obavijesti upućene određenom korisnik odnosno token-u koji je vezan za toga korisnika. Obavijesti se primaju u obliku ključ-vrijednost parova pa se za njihovo dohvaćanje koristi Map objekt kao što je prikazano u ispisu Ispis 45.

```
Map<String,String> data=remoteMessage.getData();
String title=data.get("title");
String message=data.get("message");
```

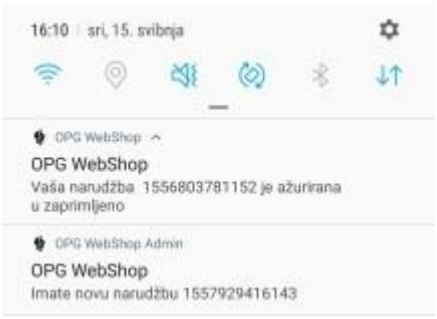
Ispis 45: Dohvaćanje obavijesti

Objekt NotificationCompat.Builder klase prikazan u ispisu Ispis 46 se koristi za kreiranje obavijesti i prikazivanje istih korisniku. Pomoću njegovih metoda postavlja se naslov i tijelo obavijesti. Također je moguće postavljanje nekih dodatnih stvari poput načina vibracije, melodije i otkazivanja obavijesti.

```
NotificationCompat.Builder builder=new NotificationCompat.Builder(this)
        .setSmallIcon(R.drawable.grape)
        .setContentTitle(title)
        .setContentText(message)
        .setAutoCancel(true)
        .setSound(defaultSoundUri)
        .setContentIntent(pendingIntent);
```

Ispis 46: Kreiranje obavijesti

Konačan rezultat odnosno izgled obavijesti je prikazan na slici Slika 20.



Slika 20: Prikaz obavijesti

5.2.2. Slanje obavijesti

FCM podržava slanje dva tipa obavijesti. Notification Message koje se prikazuju samo kad se aplikacija izvršava u pozadini i Data Message koje se prikazuju kad se aplikacija izvršava i u prednjem planu i u pozadini. Ovdje se koriste Data Message

obavijesti, a za njihovo slanje je potrebno kreirati HTTP zahtjev koji se sastoji od zaglavlja i tijela zahtjeva.

Za kreiranje i slanje zahtjeva se koristi Retrofit [7] biblioteka koja je HTTP klijent namijenjen android uređajima. Ova biblioteka zahtjeva implementaciju određenih klasa koje su potrebne za konfiguraciju Retrofit-a.

FCMRetrofitClient klasa se koristi za kreiranje i konfiguriranje Retrofit instance. Metoda unutar koje se kreira instanca je prikazana u ispisu Ispis 47.

```
public static Retrofit getClient(String baseUrl)
{
retrofit= new Retrofit.Builder().baseUrl(baseUrl)
.addConverterFactory(GsonConverterFactory.create()).build();

    return retrofit;
}
```

Ispis 47: Kreiranje Retrofit objekta

Interface klasa iz ispisa Ispis 48 se koristi za definiranje zaglavlja, tijela zahtjeva i mogućih operacija koristeći anotaciju. sendNotification metoda govori da se kao tijelo zahtjeva koristi DataMessage objekt koji predstavlja poruku koju šaljemo. Call objekt koji se nalazi ispred deklaracije sendNotification se koristi za dohvaćanje povratne vrijednosti i izvršava se prilikom svakog pozivanja sendNotification metode.

```
@Headers
        (
                {
                        "Content-Type:application/json",
                        "Authorization:key=AAAADPN5kYw:APA91"
                }
        )

@POST("fcm/send")
Call<MyResponse> sendNotification(@Body DataMessage body);
```

Ispis 48: Kreiranje Interface-a

Kada se je obavila konfiguracija Retrofit-a može se krenuti na slanje obavijesti. Obavijesti se šalju prilikom kreiranja nove narudžbe u aplikaciji za kupce ili prilikom promjene statusa narudžbe u zaprimljeno. Ovdje će se kao primjer koristiti promjena statusa (Poglavlje 4.4).

Kod promjene statusa iz baze se dohvaća narudžba kojoj se želi promijeniti status i token korisnika koji je kreirao narudžbu.

Za slanje obavijesti prikazano u ispisu Ispis 49 treba kreirati instancu Interface-a i nad tom instancom pozivati metodu sendNotification tako da joj se proslijedi DataMessage objekt koji se sastoji od poruke i token-a korisnika kojem se šalje obavijest.

```
Map<String, String> datasend=new HashMap<>();
datasend.put("title","OPG WebShop");
datasend.put("message",getResources().getString(R.string.your_order));
DataMessage dataMessage=new DataMessage(token.getToken(),datasend);
APIService mservice= FCMRetrofitClient.getClient(fcmUrl)
.create(APIService.class)
mservice.sendNotification(dataMessage).enqueue(new Callback<MyResponse>()
```

Ispis 49: Slanje obavijesti

5.3. Struktura baze podataka

Kao što je napisano u poglavlju 2.2 podaci se spremaju u uslugu za pohranu podataka Firebase u JSON formatu. Zapravo se gradi JSON stablo sa nekim glavnim čvorovima. JSON stablo se može strukturirati na dva načina. Tako da se ugnijezdi jedan čvor u drugi ili da se dubina stabla drži što plićom. U ovom slučaju se koristi flat struktura (eng. *Flat structure*) jer je modularna te je jednostavno dodavanje novih čvora. Također kod čitanja podataka se može dohvatiti samo jedan čvor, a ne cijelo stablo kao kod ugniježđene. Slika 21 prikazuje strukturu JSON stabla koje služi za spremanje i dohvaćanje podatke unutar aplikacije.

Slika 21: Struktura JSON stabla

Stablo se sastoji od 9 čvorova i svaki čvor ima svoju djecu koja su predstavljena autogeneriranim ključem. Ovi čvorovi su u aplikaciji predstavljeni modelima unutar kojih se spremaju podaci iz čvorova.

U SQL bazama podataka tablice se povezuju pomoću stranih ključeva, a kako ovo nije SQL baza bilo je potrebno pronaći način na koji povezati čvorove products i category kako bi se znalo koji proizvod pripada kojoj kategoriji. To je napravljeno tako da je u products čvor dodan par ključ-vrijednost koji predstavlja jedinstveni ključ kategorije kao što je prikazano na slici Slika 22.

Slika 22: Povezivanje čvorova category i products

Primjerice kada se klikne na određenu kategoriju u aktivnost proizvoda se proslijedi ključ kategorije i onda se radi upit nad čvorom products tako da se traže svi proizvodi koji imaju isti ključ kategorije.

6. Zaključak

Cilj rada je bila demonstracija izrade Android aplikacije koja bi ponudila novi, lakši i jednostavniji način kupnje i prodaje domaćih proizvoda. Prikazana je upotreba različitih tehnologija i implementacija najvažnijih funkcionalnosti aplikacije.

Od tehnologija treba istaknuti Firebase platformu koja se pokazala kao odličan proizvod. Korištenjem Firebase Authentication omogućena je lakša implementacija prijave i registracije korisnika. Upotreba Firebase Storage-a se također pokazala veoma korisnom zbog mogućnosti spremanja određene datoteke i dohvaćanja veze na tu datoteku.

Android platforma i Android Studio koji nudi implementaciju mnogo gotovih funkcionalnosti pokazali su se kao dobar izbor za razvoj aplikacije. Jedina zamjerka je održavanje. Samo tijekom razvoja aplikacije 5 metoda je zastarjelo te ih je trebalo zamijeniti i te funkcionalnosti ponovo implementirati.

Za izradu OPG sustava bilo je potrebno razviti dvije aplikacije namijenjene kupcima i prodavačima. Tijekom izrade rada prikazana je upotreba i izgled korisničkog sučelja koje je vrlo jednostavno za korištenje Također je prikazana i detaljna implementacija Firebase funkcionalnost koje su korištene i nekih dodatnih funkcionalnosti kao što je dohvaćanje slike iz galerije ili koristeći kameru, kreiranje skočnih izbornika, kreiranje PDF računa i implementacija RecyclerView-a.

U budućem razvoju aplikacije namijenjene kupcima bilo bi korisno dodati praćenje navika kupaca tako da im se mogu prikazati prijedlozi proizvoda na osnovu njihovih prethodni narudžbi. Kako neki kupci nemaju PayPal račun također se može uvesti mogućnost kartičnog plaćanja. S druge strane u aplikaciju namijenjenu prodavačima može se dodati praćenje statistike poput broja narudžbi i ostvarene zarade za određeni mjesec.

7. Literatura

[1] OPG, https://plaviured.hr/sto-je-opg-i-kako-ga-otvoriti/
datum zadnjeg pristupa: 02.05.2019

[2] Android OS, https://hr.wikipedia.org/wiki/Android_(operacijski_sustav)
datum zadnjeg pristupa: 03.05.2019

[3] Firebase, https://en.wikipedia.org/wiki/Firebase
datum zadnjeg pristupa: 03.05.2019

[4] Google autentifikacija, https://firebase.google.com/docs/auth/android/google-signin
datum zadnjeg pristupa: 03.05.2019

[5] PayPal Android SDK, https://github.com/paypal/PayPal-Android-SDK
datum zadnjeg pristupa: 10.04.2019

[6] ItextG Library, https://itextpdf.com/en/resources/installation-guides/installing-itext-g-android
datum zadnjeg pristupa: 15.04.2019

[7] Retrofit, https://square.github.io/retrofit/
datum zadnjeg pristupa: 20.04.2019

[8] Tržišni udio operacijskog sustava Android, http://gs.statcounter.com/os-market-share/mobile/worldwide
datum zadnjeg pristupa: 13.06.2019

[9] Slika arhitekture android sustava, https://android.tutorialhorizon.com/android-system-architecture/
datum zadnjeg pristupa: 13.06.2019