Android aplikacija za vođenje skladišta pivovare

Bahnik, Valerian

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

Stručni studij

ANDROID APLIKACIJA ZA VOĐENJE SKLADIŠTA PIVOVARE

Završni rad

Valerian Bahnik

Osijek, 2020.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac Z1S: Obrazac za imenovanje Povjerenstva za završni ispit na preddiplomskom stručnom studiju

Osijek, 05.09.2020.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za završni ispit na preddiplomskom stručnom studiju

Ime i prezime studenta:	Valerian Bahnik
Studij, smjer:	Preddiplomski stručni studij Elektrotehnika, smjer Informatika
Mat. br. studenta, godina upisa:	AI4528, 17.10.2019.
OIB studenta:	72386466586
Mentor:	Robert Šojo
Sumentor:	
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	Marina Peko
Član Povjerenstva 1:	Robert Šojo
Član Povjerenstva 2:	dr.sc. Ivana Hartmann-Tolić
Naslov završnog rada:	Android aplikacija za vođenje skladišta pivovare
Znanstvena grana rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak završnog rada	Kreirati Android aplikaciju pomoću koje se može pratiti nabava sirovina, vođenje potrošnje i statistika sirovina. Raspodjela sirovina za predefinirane vrste piva, mogućnost kreiranja nove vrste piva. Pregled inventara skladišta, mogućnost ispisa u pdf-u. Upravljanje narudžbama piva. Primjena Firebase baze podataka.
Prijedlog ocjene pismenog dijela ispita (završnog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene mentora:	05.09.2020.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

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

IZJAVA O ORIGINALNOSTI RADA

Osijek, 14.09.2020.

Ime i prezime studenta:	Valerian Bahnik
Studij:	Preddiplomski stručni studij Elektrotehnika, smjer Informatika
Mat. br. studenta, godina upisa:	AI4528, 17.10.2019.
Turnitin podudaranje [%]:	9

Ovom izjavom izjavljujem da je rad pod nazivom: **Android aplikacija za vođenje skladišta pivovare**

izrađen pod vodstvom mentora Robert Šojo

i sumentora

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

Potpis studenta:

SADRŽAJ

1. UVOD

Cilj ovog rada je složiti aplikaciju koja će omogućiti jednostavan, organiziran i pouzdan način rada u organizaciji skladišta pivovare, te njenih resursa. Kako bi se omogućio i ostvario jednostavan, organiziran i pouzdan način rada radnika u skladištu korisnik se mora registrirati u aplikaciji te s tim podacima prijavljuje u aplikaciju. Ako korisnik već ima račun, ali je zaboravio lozinku može zatražiti novu lozinku na email. Kada se korisnik prijavi u aplikaciju ima uvid u skladište svoje pivovare i može započeti s unosom i kontrolom nad sastojcima koji su dostupni u skladištu ili sastojcima koji tek moraju biti dopremljeni u skladište. Korisniku je omogućeno kreiranje prilagođenih vrsta piva kako bi se prilikom kuhanja novog piva mogli automatski oduzeti potrebni sastojci za pivo iz skladišta. Tim putem se otklanja korisnička pogreška ukoliko bi korisnik išao ručno oduzimati sastojke iz skladišta koje je odabrao za kuhanje prilagođene vrste piva. Također korisnik može primiti narudžbu klijenta. U dio s narudžbom spadaju ime klijenta, vrste piva koje klijent želi i koliko gajbi svake vrste piva. Narudžba koju je korisnik unio se može pogledati u aplikaciji ili ju korisnik može spremiti u pdf dokument i poslati nadređenom za isporuku. Aplikacija će biti povezana s online bazom podataka u koji će biti sve informacije o resursima i korisnicima koji koriste aplikaciju.

Rad je strukturiran tako da u drugom poglavlju slijedi opis svih tehnologija i radnih okruženja koje su bile potrebne za izradu. Treće poglavlje zahvaća samu funkcionalnost aplikacije. Pod funkcionalnosti (engl. *backend*) se podrazumijeva opis koda i njegova funkcija u aplikaciji. Četvrto poglavlje sadrži kako se koristi aplikacija i opis njenog dizajna (engl. *frontend*).

1.1 Zadatak završnog rada

Kreirati *Android* aplikaciju pomoću koje se može pratiti nabava sirovina, vođenje potrošnje i statistika sirovina. Raspodjela sirovina za predefinirane vrste piva, mogućnost kreiranja nove vrste piva. Pregled inventara skladišta, mogućnost ispisa u pdf-u. Upravljanje narudžbama piva. Primjena *Firebase* baze podataka.

2. OSNOVNE KOMPONENTE APLIKACIJE

U ovome poglavlju su opisane tehnologije koje su korištene za izradu projekta, te njihov kratki opis nastanka. Pod tehnologije se misli na razvojna okruženja kojima je okružen projekt, okruženja za izradu dizajna i njegove provedbe, te sigurnosni okvir kreiranja pomoćne verzije pomoću programa za verzioniranje koda.

2.1 Povijest Androida

Android je mobilni operativni sustav koji je napravio *Google*. *Android* se temelji na modificiranoj verziji Linux-a i otvoren je za sve (engl. *open source*). Glavni razlog razvoja *Androida* je bio za mobitele sa zaslonom osjetljivim na dodir. *Android* mobilni operativni sustav ne postoji samo za mobitele nego ga ima i u televizorima, automobilima i ostalim napravama koje se koriste svakodnevno.

Prema [1] *Android* je *Google* kupio 2005. godine i predstavio 2007. godine, te 2008. godine predstavio prvu inačicu *Android* sustava. Kada je *Google* izbacio *Android* 4.4 *KitKat* predstavio je "službeno" da će se svaka nova inačica *Androida* nazvati po nekom desertu. Razlog tome je što kako naglašava *Google* "ti uređaju čine naše živote slađima". No radi minimalizma i jednostavnosti od desete inačice se više neće nazivati po desertu već po broju inačice. Tako se deseta inačica naziva *Android 10*. Do dana današnjeg postoji deset inačica sustava i jedanaesta je u razvoju. Prva koja je predstavljena je (engl. *Gingerbread*) 9.2.2011. godine, a prema [6] u tablici 2.1. mogu se vidjeti sve inačice *Android* sustava. Svaka nova inačica je donijela nešto novo. *Android Nugat* je predstavio prikaz više prozora i brzi prijelaz iz jedne aplikacije u drugu, *Android Oreo* se više bazirao na vizualnu prezentaciju (engl. *User interface*) i sliku u slici (engl. *picture-in-picture*) načinu rada, itd.

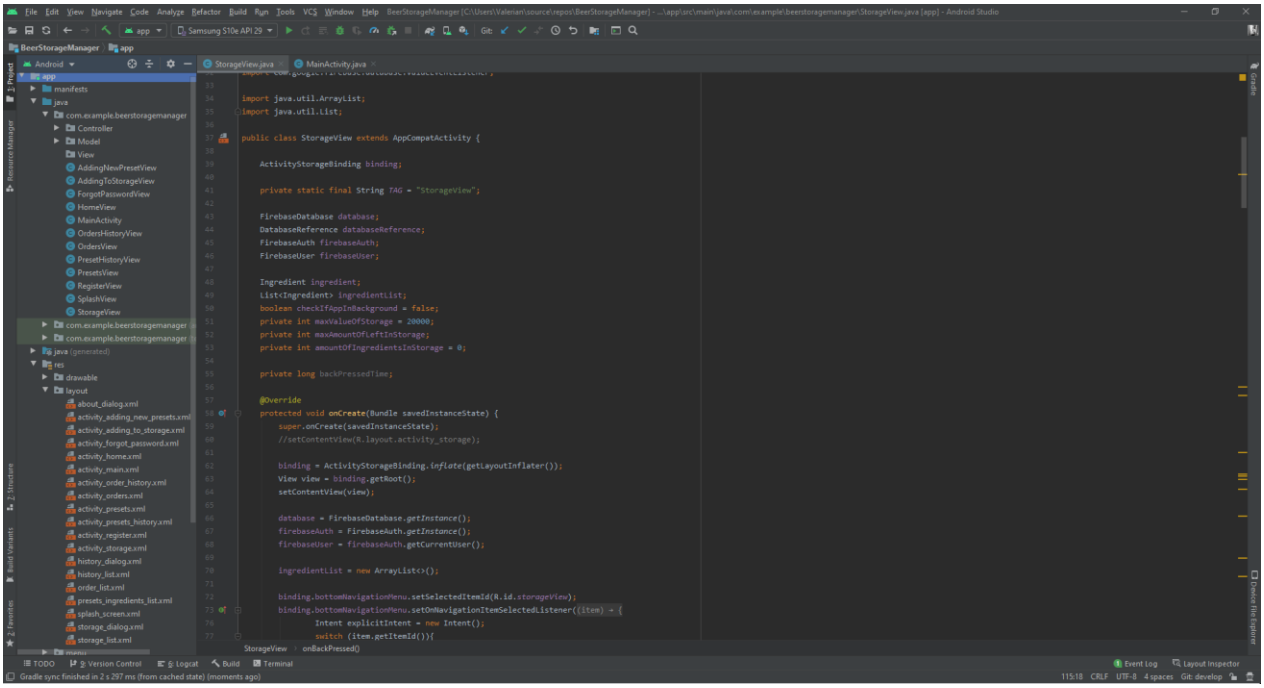
Kao i u svakom tržištu i u mobilnom postoji konkurencija. Googlovom *Android* mobilnom operativnom sustavu glavni konkurent je Appleov IOS. Dok Apple svoje uređaje prodaje po visokim cijenama i u jednom cjenovnom razredu, Google uspijeva držati veliki dio kolača tržišta sa svojom otvorenošću (engl. *open-source*) i velikim asortimanom uređaja u raznim cjenovnim razredima. Tako je *Android* postao jedan on najzastupljenijih mobilnih operativnih sustava na svijetu.

Tablica 2.1. *Android verzije.*

Verzija	Datum	API razina
Cupcake 1.5	27. travanj, 2009.	3
Donut 1.6	15. rujan, 2009.	4
Eclair 2.0 - 2.1	26. listopad, 2009.	5 – 7
Froyo 2.2 – 2.3	20. svibanj, 2010.	8
Gingerbread 2.3 – 2.3.7	06. prosinac, 2010.	9 – 10
Honeycomb 3.0 – 3.2.6	22. veljača, 2011.	11 - 13
Ice cream sandwich 4.0 – 4.0.4	18. listopad, 2011.	14 – 15
Jelly Bean 4.1 – 4.3.1	09. lipanj, 2012.	16 – 18
Kit kat 4.4 – 4.4.4	31. listopad, 2013.	19 – 20
Lollipop 5.0 – 5.1.1	12. studeni, 2014.	21 – 22
Marshmallow 6.0 – 6.0.1	05. listopad, 2015.	23
Nugat 7.0 – 7.1.2	22. kolovoz, 2016.	24 – 25
Oreo 8.0	21. kolovoz, 2017.	26 - 27
Pie 9.0	06. kolovoz, 2018.	28
10 10.0	03. rujan, 2019.	29

2.2 Android studio

Prema [2] *Android* studio je integrirano razvojno okruženje (engl. *Integrated Development Enviroment - IDE*) koje služi za izradu android aplikacija za Googlov *Android* operativni sustav. Napravljen je na temelju JetBrainsa IntelliJ IDE-a softvera dizajniran za izradu android aplikacija. Predstavljen je na Google I/O konferenciji 2013. godine. Prikaz korisničkog sučelja je na slici 2.1. Od tada je u neprestanom razvoju te ga koriste sve vrste programera, od *junior* programera pa sve do *senior* programera. Neke od važnih i vrlo dobrih mogućnosti su integrirani *Github* koji omogućava vrlo brzi i jednostavni način verzioniranja koda. Također ima brzi i konstantno nadograđivani virtualni emulator uređaja koji omogućava prikaz aplikacije u realnom vremenu. Konstantna nadogradnja, poboljšanja performansi, razvojni okvir (engl. *framework*) za testiranja itd. čine android studio prijateljskim i odličnim razvojnim okruženjem.



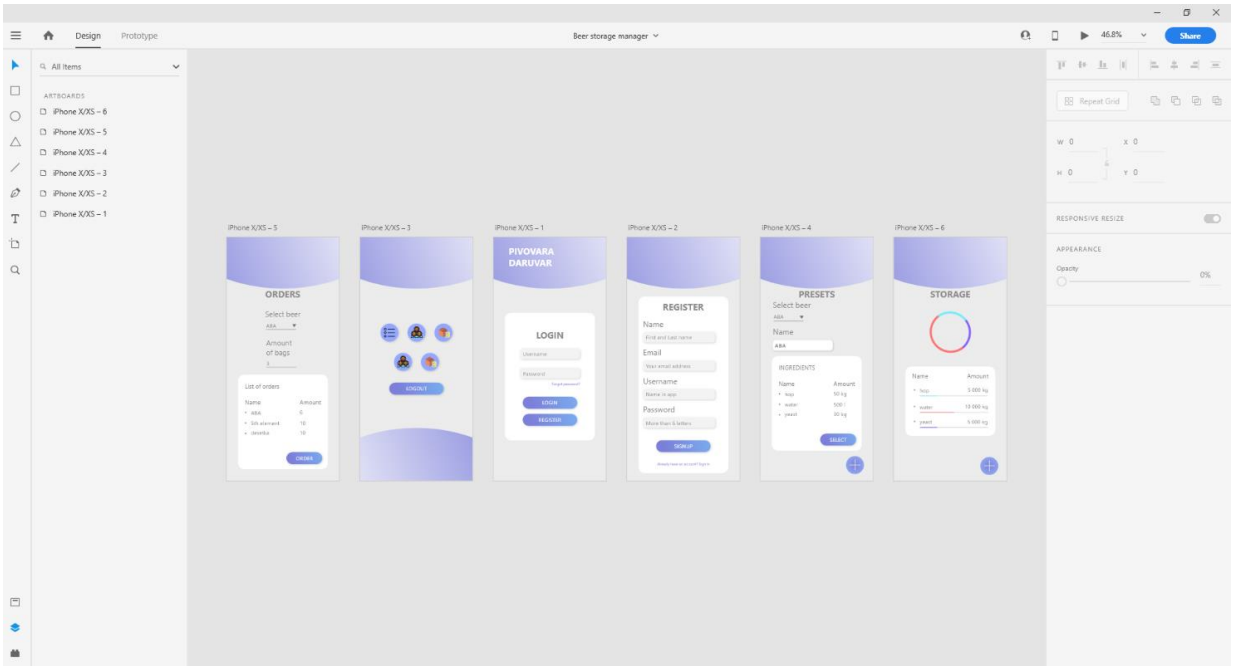
Slika 2.2. *Android studio korisničko sučelje.*

2.3 Java programski jezik

Prema [3] Java programski jezik je objektno orijentiran jezik. Razvijen u *Sun Microsystems* 1991. godine inicijalno zvan "*Oak*". Prva verzija je puštena 1995. godine nakon puno razmišljanja o imenu programa nazvan je po espresso kavi Java. Java je osmišljena tako da svako računalo ili mobitel neovisno o arhitekturi koje koristi Javu može i izvršavati aplikaciju koja je napravljena u Java programskom jeziku. Java programski jezik povlači puno sintaksi iz C i C++ programskog jezika, ali je puno jednostavnija. Java platforma se sastoji od JDK, JRE, JVM. Programeri pišu Java program u JDK (engl. *Java Development Kit*) i pokreću ga pomoću JRE (engl. *Java Runtime Environment*). Kod koji je napisan tada se prevodi preko JVM (engl. *Java Virtual Machine*) tako što se prebacuje u Java *bytecode* i tada bilo koji uređaj može uzeti taj kod i pročitati ga. Trenutno zadnja verzija Jave je 12.

2.4 Adobe XD

Prema [4] Adobe XD je relativno nova aplikacija za izradu vektorskih dizajna. Prva beta verzija je puštena na korištenje 2016. godine i originalno se zvala Adobe Experience Design CC. Vrlo je brzo postigla veliku popularnost razne publike. Velika prednost Adobe XD aplikacije naspram ostalih dizajnerskih aplikacija je velika podrška Adobea i jednostavno i prepoznatljivo sučelje. Neke od funkcionalnosti koje pruža dizajnerima su kreiranja preko govornih naredbi, engl. *drag and drop* između platna, automatska responzivnost objekata na platnu, kreiranje prototipa dizajna, kreiranje animacija za lijepi prikaz dizajna, itd. Prikaz korisničkog sučelja je na slici 2.2.



Slika 2.2. *Adobe XD korisničko sučelje.*

2.5 Firebase

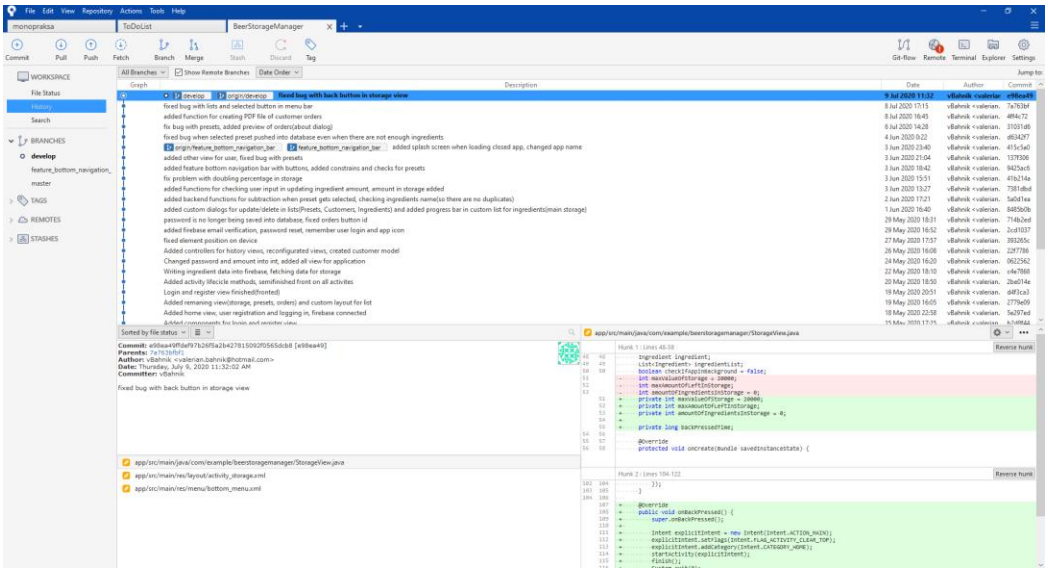
Prema [5] *Firebase* je pozadinski rad kao servis (engl. *Backend-as-a-Service*). Osnovana 2011. godine i kupljena od strane Googla 2014. godine.

Pod *Firebase* funkcionalnostima se mogu pronaći dijelovi koje inače programeri moraju raditi sami. Kako bi se mogli fokusirati na rad svoje aplikacije *Firebase* omogućava implementaciju tih funkcionalnosti kao što su analitike korisnika, autentifikacija registriranih korisnika, baza podataka, itd. Normalne baze podataka zahtijevaju od korisnika HTTP (engl. *HyperText Transfer Protocol*) upit za dohvaćanje i sinkroniziranje podataka, no kada se koristi *Firebase* tada se ne ide preko normalnog HTTP upita nego preko internet soketa (engl. *WebSocket*) koji su puno brži od HTTP upita. Dovoljan je jedan internet soket za dohvaćanje podataka sve dok to mreža može podnijeti. *Firebase* šalje promjene čim se nešto promjeni u bazi podataka, sve promjene se događa trenutno od tuda i naziv (engl. *Realtime Database*).

Firebase autentifikacija radi ono što i ime nalaže, upravlja i provjera registrirane korisnike u aplikaciji. Neke od funkcionalnosti koje bi programer morao sam implementirati su provjera valjanosti e-maila, resetiranje lozinke korisnika i mogućnost automatske prijave.

2.6 Sourcetree

Prilikom izrade aplikacije radi lakšeg snalaženja i kreiranja kompleksnih komponenti rađeno je verzioniranje koda pomoću *Git* [7] alata za verzioniranje koda. *Git* je alat koristan u izradi projekta zbog kreiranja sigurnosnih kopija. Radi lakšeg korištenja i snalaženja korišten je Sourcetree. Sourcetree je besplatan alat za grafički prikaz grananja. Ukoliko dođe do greške u aplikaciji vrlo se lako može vratiti na prethodnu verziju koja je bila valjana. U izradi je kod bio razvrstan u dvije grane - *master* i *develop*. Master grana je kreirana automatski prilikom kreiranja git *repository-a* i na njoj se nalazila samo *README* datoteka. Sve glavne komponente i funkcije koje zahvaća aplikacija su verzionirane na *develop* grani. Nakon završetka prve verzije aplikacije *develop* grana se spaja s *master* granom. U verzioniranju se moglo ići i u više grananja no u ovom projektu nije bilo potrebe. Kod više grananja se odnosi na grane posebne funkcije kao što je izrada funkcija za komunikaciju s lokalnom bazom podataka ili online bazom podataka. Takva vrta verzioniranja se koristi na većem projektu gdje sudjeluju više ljudi. Prikaz izgleda korisničkog sučelja je na slici 2.3.



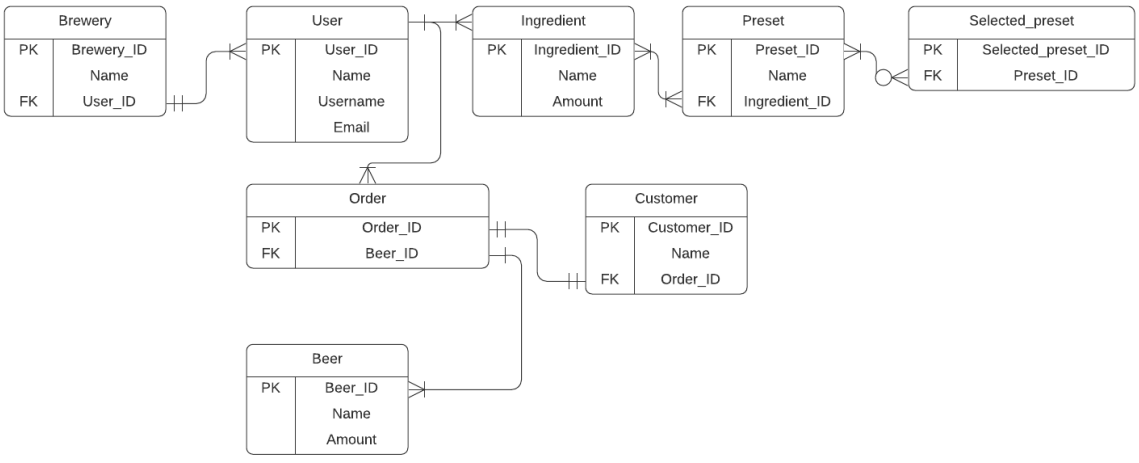
Slika 2.3. *Sourcetree korisničko sučelje.*

3. FUNKCIONALNOST APLIKACIJE

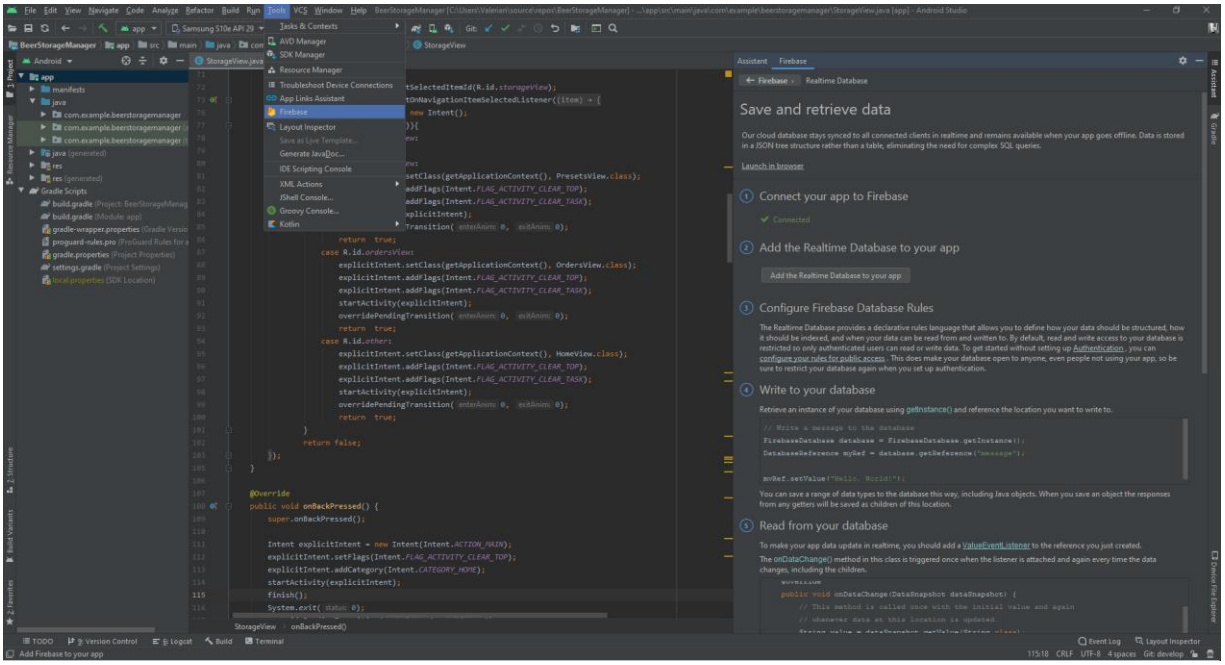
U ovom poglavlju slijedi opis funkcionalnosti aplikacije. Ukratko su opisane i prikazane glavne funkcije zadužene za komunikaciju s *Firebase* bazom podataka, provjere prilikom registracije i prijave korisnika.

3.1 Komunikacija s bazom podataka

Komunikacija s *Firebase* bazom podataka se ostvaruje postavljanje svih potrebnih biblioteka koje *Android* studio obavlja automatski prema slici 3.2. *Firebase* nudi dva rješenja pohrane podataka, jedno od rješenja je *Realtime Database* baza koja sprema podatke u JSON formatu i sinkronizira ih u realnom vremenu sa svim korisnicima, a drugo rješenje je *Cloud Firestore* koji sprema podatke u dokumente koji vrlo slični JSON formatu. Projekt ne zahtijeva veliku količinu podataka zbog svoje prirode. U skladištu pivovare se općenito ne nalazi puno sastojaka već samo oni ključni za proizvodnju piva. Tako je u ovome projektu korištena je *Realtime Database Firebase* baza podataka zbog svojih prednosti kao što je brzina sinkronizacije podataka s klijentima i jednostavnosti pohrane. *Cloud Firestore Firebase* baza podataka više ima koristi u opširnijim projektima gdje se zahtijeva kompleksnost pohrane te bi tu bilo odlično rješenje. Prikaz ER dijagrama baze podataka na slici 3.1.



Slika 3.1. *ER dijagram baze podataka.*



Slika 3.2. *Uspostava Firebasea s android projektom.*

Nakon što su biblioteke postavljene definira se objekt baze i objekt reference na bazu s kojima se pristupa bazi podataka koja je povezana s aplikacijom prema slici 3.3. i prema slici 3.4. Aplikacija je specifično povezana s jednom bazom na *Firebase* poslužitelju.

```
private void loadStorageIngredients(){
    databaseReference = database.getReference( path: "Ingredients");
    databaseReference.addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
            ingredientList.clear();
            for(DataSnapshot ingredientSnapshot : dataSnapshot.getChildren()){
                Ingredient ingredient = ingredientSnapshot.getValue(Ingredient.class);
                ingredientList.add(ingredient);
            }
        }
        @Override
        public void onCancelled(@NonNull DatabaseError databaseError) {
            displayToast("Error: database could't load.");
        }
    });
}
```

Slika 3.3. *Firebase ispis podataka iz baze.*

Slika 3.3. se odnosi na funkciju za ispis svih dostupnih sastojaka iz skladišta. Kada se pozove funkcija *loadStorageIngredients* tada objekt *databaseReference* referencira se na specifično mjesto u bazi podataka *database* koje se zove sastojci (engl. *Ingredients*). Kada je pronašao to mjesto u bazi podataka uzme sve vrijednosti iz tog mjesto koje mu je naznačeno i stavi ih u listu *ingredientList*. Nakon što su svi podaci uneseni u listu, referenca ostaje na tome mjestu i ako dođe do promjene podataka funkcija u realnome vremenu obavi radnju nad listom podataka. Ako dođe do greške prilikom povezivanja ili iščitavanja podataka javit će grešku.

```
if(!checkIfIngredientExists) {
    databaseReference = database.getReference().child("Ingredients");
    if (!TextUtils.isEmpty(name) && !TextUtils.isEmpty(amount)) {

        String id = databaseReference.push().getKey();

        ingredient = new Ingredient(id, name, amount);
        databaseReference.child(id).setValue(ingredient);
        Log.i(TAG,  msg: "Ingredient inserted into database");
    } else {
        displayToast("Ingredient is not inserted into database.");
    }
}
```

Slika 3.4. *Firebas upis podataka u bazu.*

Upisivanje u bazu podatka radi se vrlo slično kao i ispisivanje. Na slici 3.4. vidi se funkcija koja se odnosi na upis podataka u bazu *database*. Prije nego li se obavi upis u bazu radi se provjera postojanja sastojka u bazi podataka. Prema slici 3.5. ako sastojak već postoji u bazi tada se nova vrijednost unosi i zbraja s postojećom količinom sastojka u bazi. Ako sastojak koji se unosi u bazu ne postoji u bazi tada se stvara objekt sa svim vrijednostima i upisuje u bazu. Kako ne bi došlo do pogreške prilikom unosa postavljana su ograničenja koje provjeravaju da li je korisnik unio ime i količinu sastojka. Bez tih ograničenja spremali bi se nepotpuni podaci u bazu te bi moglo doći do pogreške prilikom iščitavanja.

```
private void addingNewIngredient(){
    String name = binding.addingToStorageEtIdName.getText().toString().trim();
    String amount = binding.addingToStorageEtIdAmount.getText().toString().trim();

    if(name.isEmpty() && amount.isEmpty()){
        binding.addingToStorageEtIdName.setError("Ingredient name is required");
        binding.addingToStorageEtIdName.requestFocus();
        binding.addingToStorageEtIdAmount.setError("Ingredient amount is required");
        binding.addingToStorageEtIdAmount.requestFocus();
        checkForIngredientInput = false;
        return;
    }else{
        checkForIngredientInput = true;
    }

    for (int i = 0; i < ingredientList.size(); i++) {
        Ingredient ingredientFromDatabase = ingredientList.get(i);
        if (name.equals(ingredientFromDatabase.getName())) {
            int oldStorageAmount = parseInt(ingredientFromDatabase.getAmount());
            int newStorageAmount = parseInt(amount);
            newStorageAmount += oldStorageAmount;

            databaseReference = database.getReference().child("Ingredients");
            if (!TextUtils.isEmpty(name) && !TextUtils.isEmpty(amount)) {
                ingredient = new Ingredient(ingredientFromDatabase.getIngredientId(), name, String.valueOf(newStorageAmount));
                databaseReference.child(ingredientFromDatabase.getIngredientId()).setValue(ingredient);
                checkIfIngredientExists = true;
                Log.i(TAG,  msg: "Ingredient amount value updated");
            } else {
                displayToast("Error: ingredient amount is not updated");
            }
        }
    }
```

Slika 3.5. *Provjera postojanosti sastojaka u skladištu.*

Nakon što su svi podaci ispravno uneseni u bazu i provjereni, funkcija *listingIngredients* prema slici 3.6. ispisuje sve sastojke koji se nalaze u bazi podataka. Kako bi se ispisali podaci funkcija prvo uspostavi vezu s bazom na mjestu u bazi gdje se sastojci nalaze. Zatim uzima listu *ingredientList* i učitava sve sastojke u listu. Za prikaz ukupne količine sastojaka i njezinih postotak uzimaju se sve količine pojedinih sastojaka te preračunavaju u varijablu *percentage* koju se postavlja za prikaz na *storageTvIdPercentage* tekst.

```
private void listingIngredients(){

    databaseReference = database.getReference( path: "Ingredients");
    databaseReference.addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
            ingredientList.clear();
            for(DataSnapshot ingredientSnapshot : dataSnapshot.getChildren()){
                Ingredient ingredient = ingredientSnapshot.getValue(Ingredient.class);
                ingredientList.add(ingredient);

                int amount = Integer.parseInt(ingredient.getAmount());
                amountOfIngredientsInStorage += amount;
                if (!checkIfAppInBackground) {
                    int percentage;
                    percentage = amountOfIngredientsInStorage * 100 / maxValueOfStorage;
                    binding.storageTvIdPercentage.setText(percentage + "%");
                    maxAmountOfLeftInStorage = maxValueOfStorage - amountOfIngredientsInStorage;
                    binding.storageProgressBarIdProgressBar.setMax(maxValueOfStorage);
                    binding.storageProgressBarIdProgressBar.setProgress(amountOfIngredientsInStorage);
                }
            }
            IngredientListController adapter = new IngredientListController( context: StorageView.this, ingredientList, maxValueOfStorage);
            binding.storageListIdBeers.setAdapter(adapter);
        }
        @Override
        public void onCancelled(@NonNull DatabaseError databaseError) {
                displayToast("Error: database could't load.");
        }
    });
```

Slika 3.6. *Funkcija za prikaz podataka.*

Za jednostavan i minimalistički prikaz brine se *IngredientListController*. Prema slici 3.6. može se vidjeti kako adapter dobiva listu *ingredientList* i maksimalnu vrijednost količine sastojaka u skladištu *maxValueOfStorage* od podataka. Daljnja manipulacija tim podacima prikazana je na slici 3.7.

```java
public class IngredientListController extends ArrayAdapter<Ingredient> {

    private Activity context;
    private List<Ingredient> IngredientList;
    private int maxAmount;

    public IngredientListController(Activity context, List<Ingredient> ingredientList, int max){
        super(context, R.layout.storage_list, ingredientList);
        this.context = context;
        this.IngredientList = ingredientList;
        maxAmount = max;
    }
    public IngredientListController(Activity context, List<Ingredient> ingredientList){
        super(context, R.layout.storage_list, ingredientList);
        this.context = context;
        this.IngredientList = ingredientList;
    }

    @NonNull
    @Override
    public View getView(int position, @Nullable View convertView, @NonNull ViewGroup parent) {
        LayoutInflater inflater = context.getLayoutInflater();

        View ingredientViewItem = inflater.inflate(R.layout.storage_list,  root: null,  attachToRoot: true);

        TextView tvName = (TextView) ingredientViewItem.findViewById(R.id.storage_list_tvIdBeerName);
        TextView tvAmount = (TextView) ingredientViewItem.findViewById(R.id.storage_list_tvIdBeerAmount);
        ProgressBar progressBar = (ProgressBar) ingredientViewItem.findViewById(R.id.storage_list_pbId);

        Ingredient ingredient = IngredientList.get(position);

        tvName.setText(ingredient.getName());
        tvAmount.setText(String.valueOf(ingredient.getAmount()));

        progressBar.setMax(maxAmount);
        progressBar.setProgress(Integer.valueOf(ingredient.getAmount()));

        return ingredientViewItem;
    }
}
```

***Slika 3.7**. Adapter za prikaz podataka.*

Na slici 3.7. može se vidjeti izgled *IngredientListController* adapter za prikaz podataka. Funkcija *IngredientListController* prima podatke koji su uneseni u funkciju na slici 3.6. *IngredientListController*. Podaci koje prima su *activity* u kojem se prikazuje, lista sa sastojcima i maksimalnu vrijednost. Kada su svi podaci dohvaćeni pomoću funkcije spremaju se u lokalne varijable i liste kako bi se mogli koristiti za postavljanje prikaza u klasi. Lista sastojaka sprema se u *IngredientList* listu, a maksimalna vrijednost tj. količina sastojaka u skladištu u varijablu *maxAmount*.

Te vrijednosti se prosljeđuju u funkciju *View* koja zatim postavlja podatke za prikaz u tekstualne prikazu *tvName*, *tvAmount* i *progressBar* koji predstavlja kružni prikaz količine podataka u skladištu.

3.2 Registracija korisnika

Uspostava registracija korisnika može biti vrlo zahtijevan i dugotrajan dio projekta. Kako bi se osigurala i olakšana registracija postavljene su sigurnosne provjere. Na daj način je riješen problem registriranja korisnika s nepotpunim ili netočnim podacima. Kao što su krivo unesene e-mail adrese i prazna polja podataka. Provjera su vidljive na slici 3.8.

```
private void RegisterUser(){
    String name = binding.registerEtIdName.getText().toString().trim();
    String email = binding.registerEtIdEmail.getText().toString().trim();
    String username = binding.registerEtIdUsername.getText().toString().trim();
    String password = binding.registerEtIdPassword.getText().toString().trim();

    if(name.isEmpty()){
        binding.registerEtIdName.setError("Name is required");
        binding.registerEtIdName.requestFocus();
        return;
    }
    if(email.isEmpty()){
        binding.registerEtIdEmail.setError("Email is required");
        binding.registerEtIdEmail.requestFocus();
        return;
    }
    if(username.isEmpty()){
        binding.registerEtIdUsername.setError("Username is required");
        binding.registerEtIdUsername.requestFocus();
        return;
    }
    if(password.isEmpty()){
        binding.registerEtIdPassword.setError("Password is required");
        binding.registerEtIdPassword.requestFocus();
        return;
    }
    if (!Patterns.EMAIL_ADDRESS.matcher(email).matches()){
        binding.registerEtIdEmail.setError("Please enter a valid email");
        binding.registerEtIdEmail.requestFocus();
        return;
    }
    if(password.length() <= 6){
        binding.registerEtIdPassword.setError("Minimum length of password should be 6");
        binding.registerEtIdPassword.requestFocus();
        return;
    }
```

Slika 3.28. *Provjere ispravnosti podataka.*

Firebase authentication SDK je biblioteka koju omogućava *Firebase* za jednostavnu implementaciju sigurnosne autentifikacije registriranih korisnika, registriranje korisnika i resetiranje lozinke. Funkcija zadužena za kreiranja novog korisnika pomoću *Firebase authentication* je *createUserWithEmailAndPassword* što se može vidjeti na slici 3.9. Uzima korisnikovu e-mail adresu i lozinku te kreira korisnika.

Kada funkcija kreira novog korisnika pošalje e-mail na e-mail koji je korisnik postavio kako bi se potvrdila njegova valjanost. Tek tada se e-mail potvrdi i dokaže njegova postojanost, te se korisnik može prijaviti u aplikaciju. Ako korisnik prilikom registracije postavi e-mail koji je već registriran pojavit će se poruka da taj račun već postoji.

```
firebaseAuth.createUserWithEmailAndPassword(email, password).addOnCompleteListener(new OnCompleteListener<AuthResult>() {
    @Override
    public void onComplete(@NonNull Task<AuthResult> task) {
        if(task.isSuccessful()){
            FirebaseUser firebaseUser = firebaseAuth.getCurrentUser();

            firebaseUser.sendEmailVerification().addOnCompleteListener((task) -> {
                    if(task.isSuccessful()){
                        displayToast("Registered successfully. Please verify your email address");
                        Intent explicitIntent = new Intent();
                        explicitIntent.setClass(getApplicationContext(), MainActivity.class);
                        startActivity(explicitIntent);
                    }else {
                        displayToast(task.getException().getMessage());
                    }
            });
        }else{
            if(task.getException() instanceof FirebaseAuthUserCollisionException){
                displayToast("This account already exist.");
            }else{
                displayToast(task.getException().getMessage());
            }
        }
    }
});
```

Slika 3.9. *Firebase authentication.*

Prilikom registracije korisnik unosi svoje podatke koji se spremaju u bazu podataka *database* tako što se *databaseReference* referencira na specifično mjesto u bazi koje se naziva korisnik (engl. *User*) i sprema novo kreiranog korisnika s unikatnim identifikacijskim ključem (engl. *ID*). Razlog spremanja korisnikovih podataka jest radi evidencije i mogućeg raspoređivanja u različite pivovare.

Slika 3.10. *Spremanje novog korisnika u bazu podataka.*

3.3 Prijava korisnika

Nakon što je novi korisnik registriran i potvrđena valjanost e-mail adrese može se prijaviti i započeti rad u aplikaciji. Metoda koja je zadužena za provjeru unesenih podataka *loginUser* provjerava nalazi li se korisnik u bazi podataka. Na slici 3.11. može se vidjeti kako korisnik mora unijeti e-mail i lozinku koju je unio prilikom registracije, te se vrše provjere kako se ne bi dogodilo da se prijavi korisnik samo s lozinkom. U *Firebase authentication* biblioteci funkcija za provjeru registriranih korisnika *signInWithEmailAndPassword* uzima unesenu lozinku i e-mail te provjerava da li su valjani uneseni podaci i je li e-mail potvrđen. Ukoliko su podaci valjani dozvoljava se korisniku prijava.

```
private  void loginUser(){
    String email = binding.loginEtIdEmail.getText().toString().trim();
    String password = binding.loginEtIdPassword.getText().toString().trim();

    if(email.isEmpty()){
        binding.loginEtIdEmail.setError("Email is required");
        binding.loginEtIdEmail.requestFocus();
        return;
    }
    if(password.isEmpty()){
        binding.loginEtIdPassword.setError("Password is required");
        binding.loginEtIdPassword.requestFocus();
        return;
    }

    firebaseAuth.signInWithEmailAndPassword(email, password).addOnCompleteListener((task) → {
            if(task.isSuccessful()){
                FirebaseUser firebaseUser = firebaseAuth.getCurrentUser();
                if(firebaseUser.isEmailVerified()){
                    startActivity(new Intent(getApplicationContext(), StorageView.class));
                    displayToast("Logged in");
                }else{
                    displayToast("Please verify your email address");
                }
            }else{
                displayToast(task.getException().getMessage());
            }
    });
}
```

Slika 3.11. *Funkcija za provjeru postojanja korisnika.*

3.4 Kreiranje narudžbe

Kreiranje nove narudžbe odvija se u *OrderView* klasi. Kako bi korisnik kreirao novu narudžbu prvo se unosi naziv klijenata prema slici 3.12. Funkcija *orderForCustomer* uzima naziv klijenta koji je unesen u varijablu *CustomerName*, ako je varijabla prazna i korisnik zatraži aplikaciju za sljedeći korak ispisat će mu grešku i postaviti korisnika na dio gdje nije pravilno unesen podatak. Ako je ime klijenta uneseno pravilno otvara se veza s bazom podataka na lokaciji *Customers* gdje se spremaju nazivi klijenata s njihovom narudžbom.

```
private boolean orderForCustomer(){
    String CustomerName = binding.ordersEtIdOrderName.getText().toString().trim();

    if(CustomerName.isEmpty()){
        binding.ordersEtIdOrderName.setError("Customer name is required");
        binding.ordersEtIdOrderName.requestFocus();
        checkForCustomerName = false;
    }else{
        checkForCustomerName = true;
    }

    databaseReference = database.getReference().child("Customers");
    if(!TextUtils.isEmpty(CustomerName)){

        String id = databaseReference.push().getKey();

        customer = new Customer(id, CustomerName, orderId);
        databaseReference.child(id).setValue(customer);
        Log.i(TAG,  msg: "Customer inserted into database");
    } else {
        Log.i(TAG,  msg: "Customer is not inserted into database.");
    }
    return checkForCustomerName;
}
```

Slika 3.12. *Funkcija za unos klijenta.*

Pravilnim unosom naziva klijenta korisnik unosi naziva piva i količine piva koju klijent naručuje. Za unos podataka o narudžbi brine se funkcija *addingItemsInNewOrder* prema slici 3.13. U varijable *BeerName* i *amount* spremaju se naziv piva i količina koje zatim prolaze kroz provjeru, ako su neispravno uneseni podaci zatražit će korisnika ponovni unos. Svako pivo koju unese mora zasebno potvrditi kako ne bi došlo do pogreške prilikom unosa u privremenu listu *beerOrderList* vidljivu na slici 3.14. koja služi za prikaz piva koje je klijent naručio. Potvrda se odvija pritiskom na dugme input. Ako je krivo uneseno pivo dugim pritiskom na naziv piva otvara se dijalog koji nudi opciju za otklanjanje piva iz liste. Kada je narudžba upotpunjena uspostavlja se veza s bazom podataka i unose podaci u bazu na mjesto *Orders* pod stranim ključem koji pripada nazivu klijenta.

```
private void addingItemInNewOrder(){

    String BeerName = binding.ordersEtIdBeerName.getText().toString().trim();
    String amount = binding.ordersEtIdAmountOfBags.getText().toString();

    if(BeerName.isEmpty() && amount.isEmpty()){
        binding.ordersEtIdBeerName.setError("Beer name is required");
        binding.ordersEtIdBeerName.requestFocus();
        binding.ordersEtIdAmountOfBags.setError("Amount of bags is required");
        binding.ordersEtIdAmountOfBags.requestFocus();
        checkForOneOrder = false;
        checkForCustomerName = false;
        return;
    }else{
        checkForOneOrder = true;
        checkForCustomerName = true;
    }

    databaseReference = database.getReference().child("Orders");
    if(!TextUtils.isEmpty(BeerName)){

        String id = databaseReference.push().getKey();

        beer = new Beer(id, BeerName, amount);
        databaseReference.child(orderId).child(id).setValue(beer);
        Log.i(TAG,  msg: "Beer inserted into database");
        binding.ordersEtIdBeerName.getText().clear();
        binding.ordersEtIdAmountOfBags.getText().clear();
    } else {
        Log.i(TAG,  msg: "Beer is not inserted into database.");
    }
}
```

Slika 3.13. *Funkcija za unos podataka o narudžbi.*

Sva piva koja su bila unesena u listu za trenutni prikaz prije samog postavljana u bazu te završetka narudžbe se trenutno prikazuju pomoću funkcije *displayInputtedOrder* prema slici 3.14.

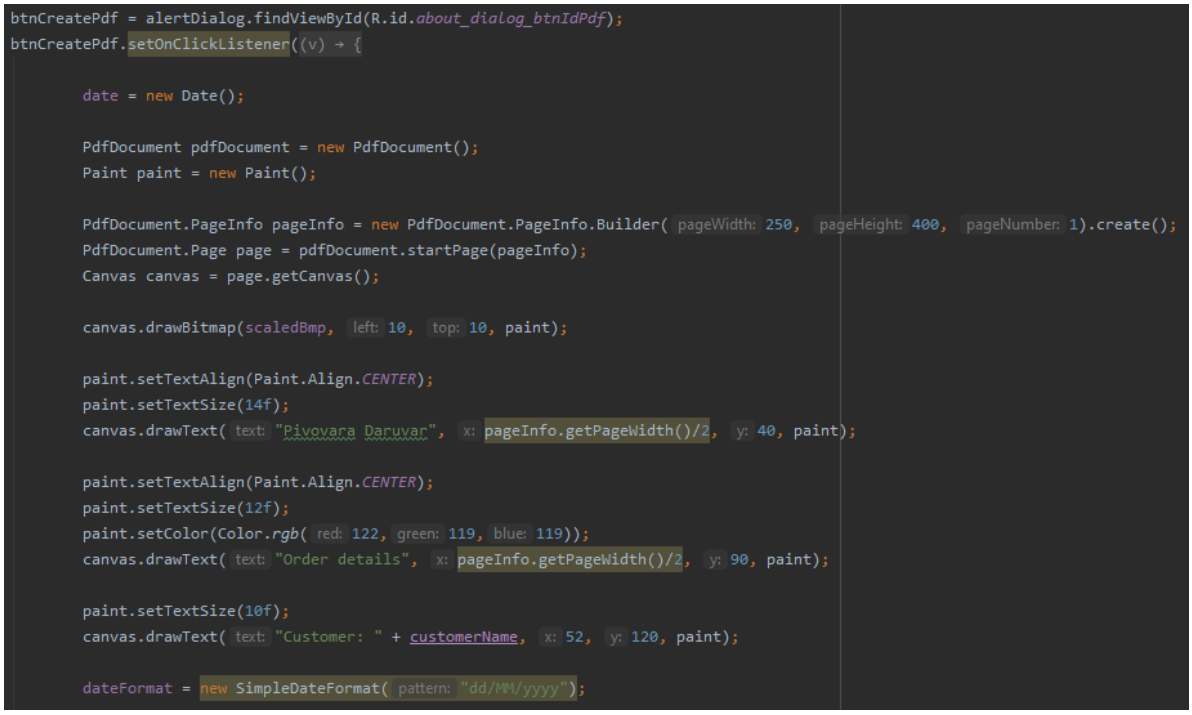
```
private void displayInputtedOrder(){
    databaseReference = database.getReference( path: "Orders").child(orderId);
    databaseReference.addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
            beerOrderList.clear();
            for(DataSnapshot ingredientSnapshot : dataSnapshot.getChildren()){
                Beer beer = ingredientSnapshot.getValue(Beer.class);
                beerOrderList.add(beer);
            }
            OrdersListController adapter = new OrdersListController( context: OrdersView.this, beerOrderList);
            binding.ordersListIdBeers.setAdapter(adapter);
        }
        @Override
        public void onCancelled(@NonNull DatabaseError databaseError) {
            displayToast("Error: database could't load.");
        }
    });

    binding.ordersListIdBeers.setOnItemClickListener((parent, view, position, id) -> {
            beer =  beerOrderList.get(position);
            deleteDialog(beer.getBeerId());
    });

}
```

Slika 3.14. *Funkcija za trenutni prikaz podataka.*

Funkcija *displayInputtedOrder* dobiva referencu na mjesto u bazi gdje su spremljene trenutno spremljena piva. Piva sprema u *beerOrderList* listu te ih ispisuje pomoću adapter *OrderListController*.

3.5 Kreiranje pdf formata

Nakon uspješno kreirane narudžbe koju je klijent zatražio korisnik ima mogućnost spremiti narudžbu u pdf format kako bi ju proslijedio zaduženom osoblju u pivovari za slanje narudžbe. Lista u koju se dohvaćaju narudžbe iz baze podataka popunjava se na isti način kao i sve prethodne liste. Uspostavlja se veza s bazom podataka na specifično mjesto te se popuni lista podacima. Uspostavljanje veze s bazom može se vidjeti na slici 3.14. gdje je *database* objekt *FirebaseDatabase* i *databaseReference* objekt *DatabaseReference*. U tom slučaju *database* je objekt koji predstavlja bazu podataka, a *databaseReference* je putanja do podataka tj. referenca.



Slika 3.15. *Kreiranje pdf formata.*

Kreiranje pdf formata odvija se u dijalogu koji se otvara kada korisnik pritisne naziv klijenta, te gdje pritiskom na dugme *btnCreatePdf* se kreira posebno izrađena pdf datoteka. Posebno kreirana pdf datoteka može se vidjeti na slikama 3.15. i 3.16. Za stvaranje pdf formata zadužene su ugrađene android funkcije *PdfDocument*, *Paint* i *Canvas*. Funkcija *PdfDocument* stvara novi pdf objekt *pdfDocument* što predstavlja novu pdf datoteku zatim funkcija *Paint* na koju se radi objekt *paint* koji služi za postavljanje pozicije elemenata na praznome pdf platnu. Prije nego li se započne postavljanje elemenata u pdf kao što su naslov, datum i ostali tekst, *PdfDocument* poziva svoje metode za postavljanje osnovne informacije o pdf-u. *PdfDocument.PageInfo* je metoda koja kreira objekt *pageInfo* koji postavlja širinu, dužinu i broj stranica, te kreira pdf s tim karakteristikama. Zatim *PdfDocument.Page* postavlja objekt *page* koji govori na kojoj stanici započinje pdf dokument. Kao zadnji korak kod kreiranja pdf dokumenta funkcija *Canvas* kreira objekt *canvas* koji postavlja platno po kojem se budu postavljali elementi pdf-a.

Kada su svi koraci kreiranja postavljeni započinje unos elemenata na platno. Prvo se postavlja karakteristika teksta koji se unosi vidljivo na slici 3.15. Pozivom *paint.setTextAlign(Paint.Align.CENTER)* tekst se centrira na sredinu platna, a *paint.setTextSize(14f)* određuje se veličina fonta teksta. Nakon uspostavljeni karakteristika teksta poziva se *canvas.drawText()* koji postavlja tekst koji je unesen, te postavlja *padding* oko teksta kako je određeno.

```
paint.setTextSize(10f);
canvas.drawText( text: "Date: " + dateFormat.format(date),  x: 48,  y: 140, paint);

paint.setTextSize(12f);
canvas.drawText( text: "Beer name",  x: 55,  y: 170, paint);

canvas.drawLine( startX: 25,  startY: 174,  stopX: 210,  stopY: 174, paint);

paint.setTextSize(12f);
canvas.drawText( text: "Quantity",  x: 180,  y: 170, paint);

int nameStartXPosition = 65;
int amountStartXPosition = 180;
int startYPosition = 195;

for(int i = 0; i < beerList.size(); i++){
    Beer beer = beerList.get(i);
    canvas.drawText(beer.getName(), nameStartXPosition, startYPosition, paint);
    canvas.drawText(beer.getAmountOfBags(), amountStartXPosition, startYPosition, paint);
    startYPosition += 20;
}


pdfDocument.finishPage(page);

File file = new File(Environment.getExternalStorageDirectory(),  child: "/"+customerName + " Order Details.pdf");

try {
    pdfDocument.writeTo(new FileOutputStream(file));
    displayToast("PDF created");
} catch (IOException e) {
    e.printStackTrace();
    displayToast("Error: PDF not created");
}
pdfDocument.close();
```

Slika 3.16. *Završni dio izrade pdf formata.*

Nakon što su svi elementi uneseni u funkciju za kreiranje pdf dokumenta na samome kraju nalazi se funkcija *pdfDocument.finishPage(page)* koja označava kraj pdf stranice. Zatim se poziva funkcija *File* koja kreira objekt *file* koji predstavlja novu datoteku koju će pokušati spremiti u unutarnju memoriju uređaja. Svaka nova datoteka ima svoji naziv *customerName* kao što je vidljivo na slici 3.16. Ako je datoteka uspješno kreirana ispisat će se poruka "PDF created" kako bi korisnik znao da je datoteka uspješno kreirana u suprotnom ispisat će se poruka "Error: PDF not created". Završni korak kod kreiranja pdf dokumenta je funkcija *pdfDocument.close()* koja zatvara pdf koji je u izradi.

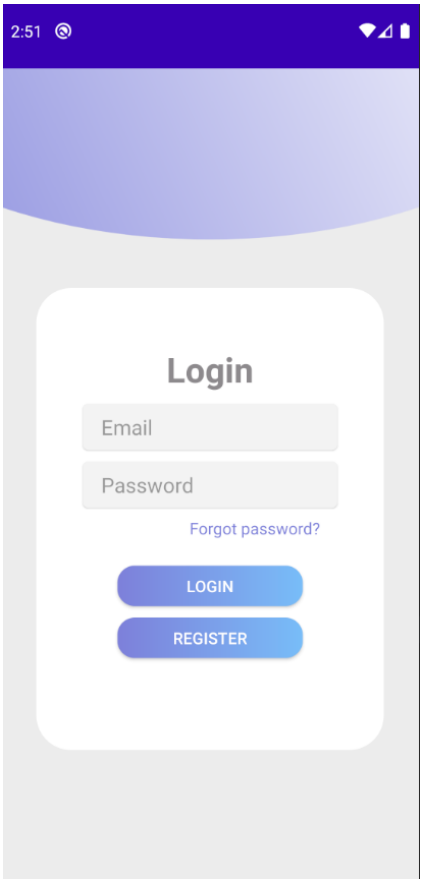
4. UPUTE ZA KORIŠTENJE APLIKACIJE I TESTIRANJE

U ovome poglavlju bit će opisan način rada aplikacije. Svaki od koraka ima slikovni prikaz i opis što se nalazi na slici. Prvi korak je registracija korisnika i prijava, te ostale mogućnosti. Drugi korak je prijava i upoznavanje sučelja. U kratkim crtama su opisani dijelovi aplikacije i čemu služe.

4.1 Registracija korisnika

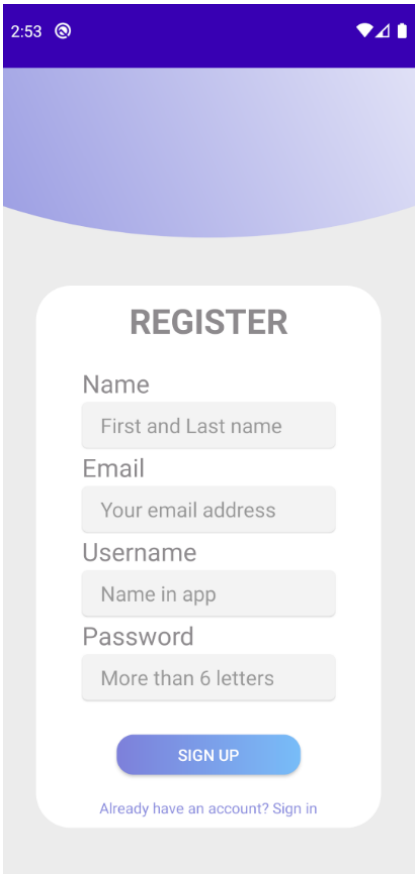
Kako bi se započeo rad u aplikaciji korisnik mora kreirati svoj novi račun. Na slici 4.1. prikazan je početni zaslon za korisnika koji nije prijavljen. Kako bi se novi račun kreirao korisnik mora pritisnuti na dugme *REGISTER*.

Nakon pritisnutog dugmeta *REGISTER* korisnika aplikacija vodi na sljedeći prozor koji traži od korisnika da unese svoje podatke. Na slici 4.2. vidljivi su svi podaci koji se zahtijevaju od korisnika. Kada su sva vidljiva polja popunjena i ispravno unesena pritiskom na dugme *SIGN UP* izvršava se registracija.



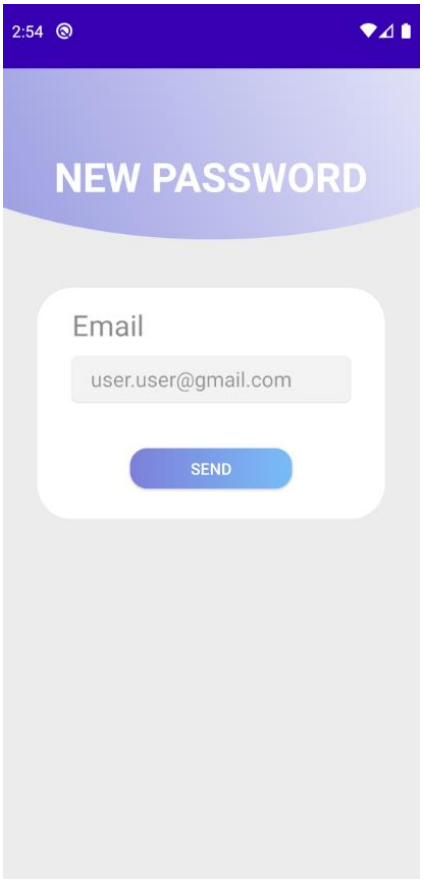
Slika 4.1. *Početni zaslon.*

Ako su valjani podaci šalje se e-mail za potvrdu ispravnosti unesenog e-maila, te kako bi se omogućila prijava korisnik mora otići na e-mail i potvrditi ga. Ako je korisnik slučajno otvorio registracijski prozor može pritisnuti tekst ispod dugmeta *SIGN UP* kako bi se vratio na zaslon za prijavu.



Slika 4.2. *Prikaz registracijskog zaslona.*

Kada se korisnik vratio na zaslon na prijavu i potvrdio e-mail adresu vraćen je na zaslon na prijavu koji je vidljiv na slici 4.1. Za prijavu u aplikaciju potrebno je popuniti polja podacima koji se zahtijevaju. Ako je korisnik prilikom prijave zaboravio lozinku može pritisnuti tekst *Forgot password?* koji će ga odvesti na zaslon zadužen za zatražene nove lozinke. Na slici 4.3. vidi se zaslon za zahtjev nove lozinke. Kako bi aplikacija poslala zahtjev za novu lozinku korisnik mora unijeti svoju e-mail adresu, te će na e-mail dobiti upute kako postaviti novu lozinku.

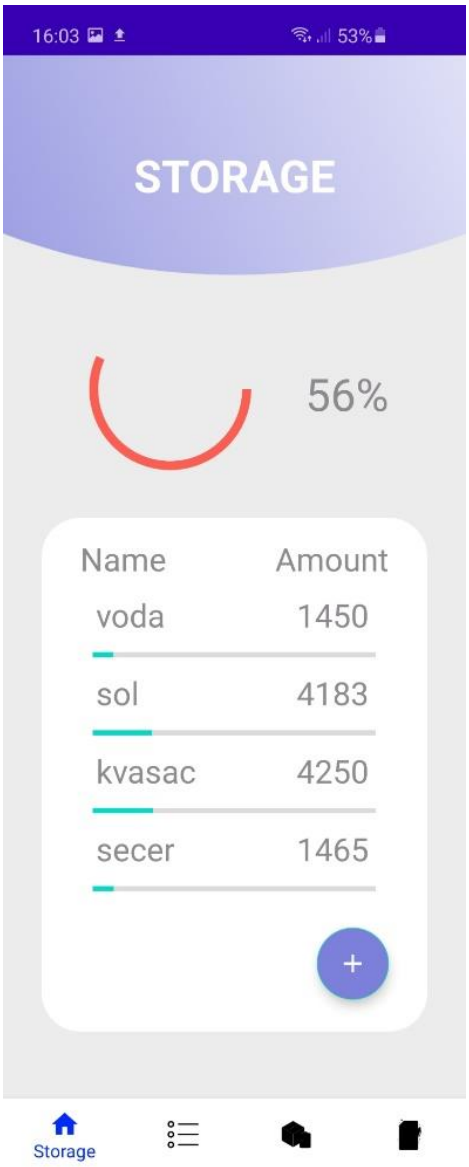


Slika 4.3. *Zaslon za zahtijev nove lozinke.*

4.2 Upravljanje skladištem

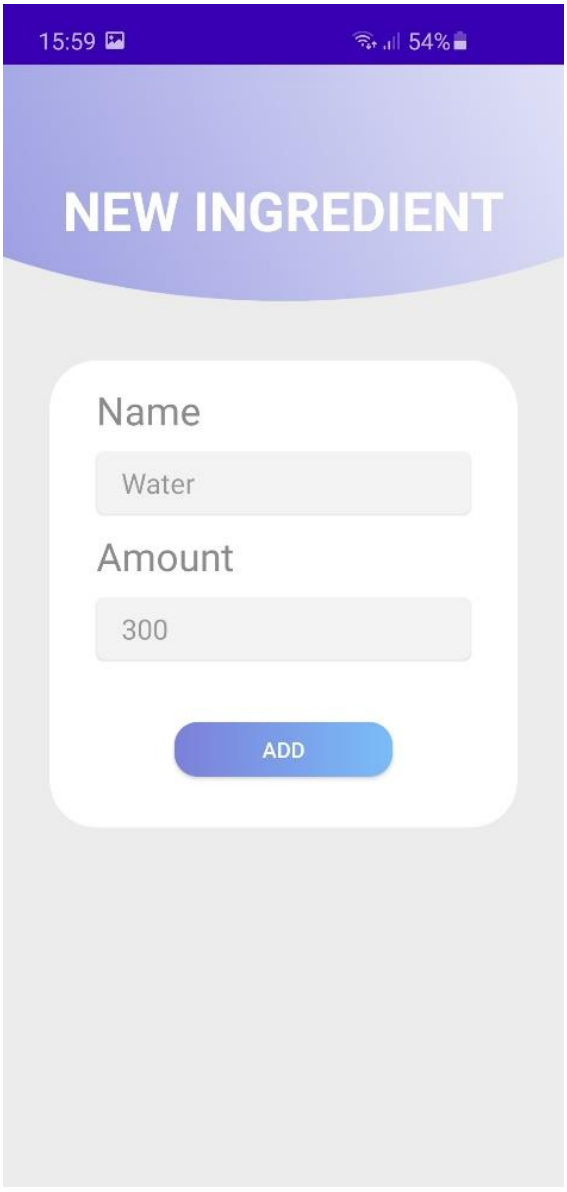
Pritiskom na dugme *LOGIN* otvara se početni zaslon *STORAGE* koji će se uvijek otvoriti prvi ukoliko se korisnik nije odjavio iz aplikacije. Na slici 4.4. može se vidjeti sastav *STORAGE* zaslona. Na ovom zaslonu obavlja se sva statistika vezana za glavno skladište, kao što je upravljanje sirovinama koje se nalaze u skladištu.

Dolaskom novih sastojaka u skladište njihov unos bi se odvijao pod zaslonom za unos novih sastojaka. Pritiskom na dugme s oznakom plus otvara se zaslon za unos novog sastojka kao što je prikazano na slici 4.5.



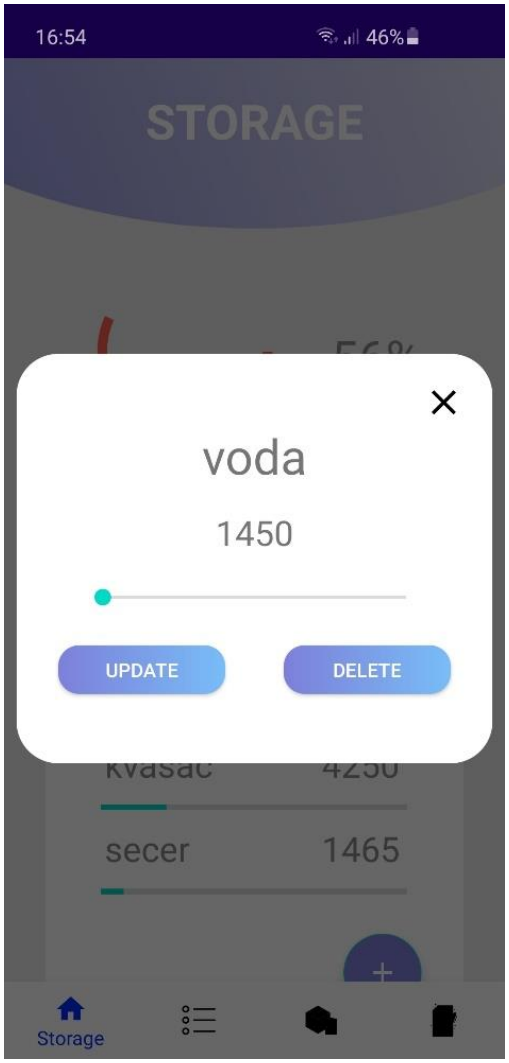
Slika 4.4. *Storage zaslon.*

Nakon otvaranja zaslona za unos novog sastojka korisnik unosi podatke o sastojku i pritiskom na gumb *ADD* unosi ga u skladište.



Slika 4.5. *Zaslon za unos novog sastojka.*

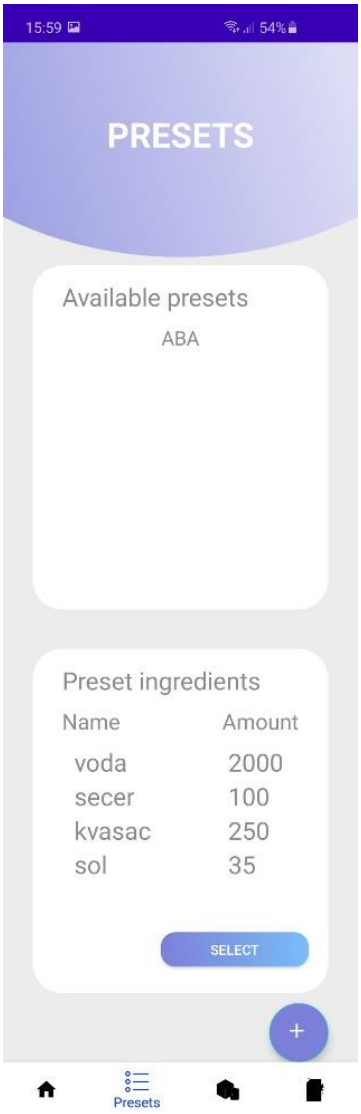
Ako sastojak kojeg korisnik unosi postoji u skladištu nadodat će na taj isti sastojak novu količinu sastojka. Ako je krivo unesena veličina korisnik može pritiskom na sastojak otvoriti dijalog koji će mu omogućiti promjenu vrijednosti pomoću kliznog odabira kao što je prikazano na slici 4.6.



Slika 4.6. *Dijalog za mijenjanje vrijednosti.*

4.3 Odabir predefinirane vrsta piva

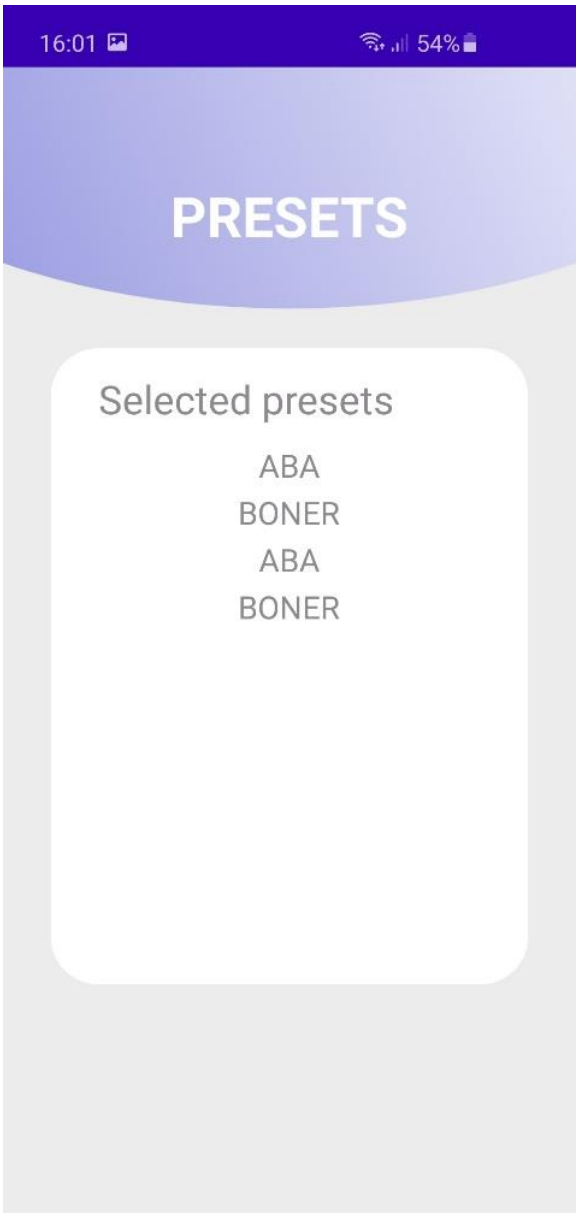
Prema slici 4.4. može se vidjeti navigacijska komponenta na dnu zaslona. Ona služi za jednostavnu navigaciju kroz aplikaciju. Kako bi korisnik otvorio zaslon za odabir predefinirane vrste piva mora odabrati ikonu nalik na listu koja je druga po redu s lijeva na desno. Lista predstavlja *PRESETS* zaslon u kojem se nalaze sve predefinirane vrste piva. Na slici 4.7. prikaz *PRESETS* zaslona.



Slika 4.37. *Presets zaslon.*

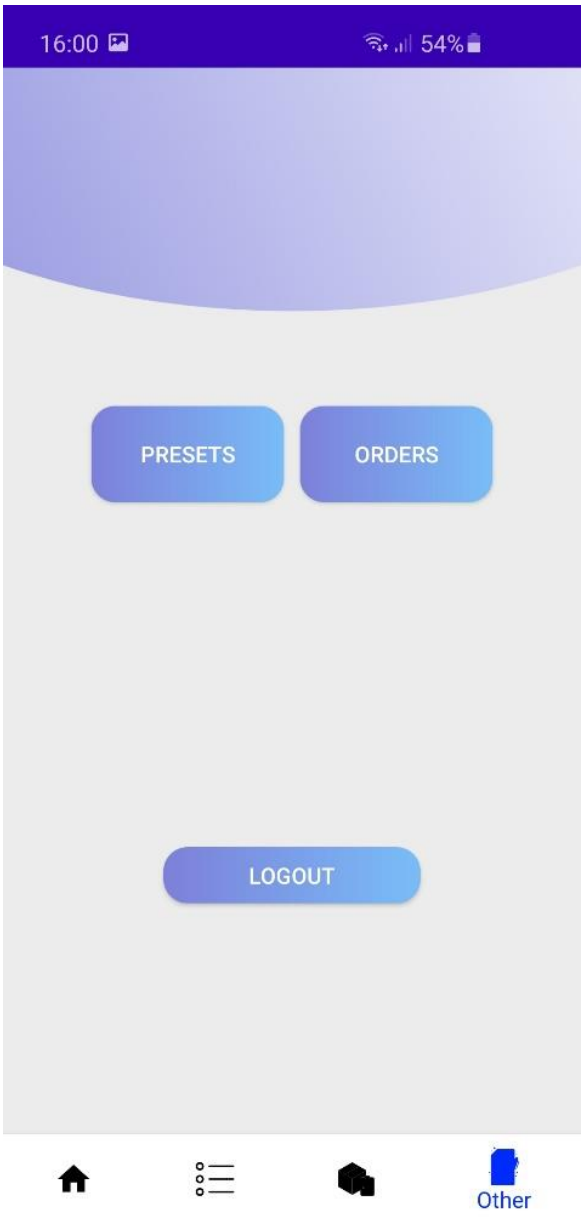
Pritiskom na ime pod listom predefiniranih vrsta piva, prikazuju se sastojci potrebni za proizvodnju tog piva. Kada bi u pivovari krenuli proizvoditi neko pivo korisnik bi odabrao vrstu i pritiskom na dugme *SELECT* odabire se odabrana vrsta piva. Aplikacija zatim provjerava nalazi li se u skladištu dovoljno sastojaka za kuhanje te vrste piva. Ako nema dovoljno sastojaka javlja kako se u skladištu ne nalazi dovoljno sastojaka za proizvodnju tog piva, a ako ima tada se ime tog piva sprema u listu gdje se može pogledati koje su sve vrste piva bile u proizvodnji i otvara zaslon s prethodno odabranim vrstama piva. Lista svih piva koje su se proizvodile nalaze se u zaslon s odabranim vrstama *PRESETS* prema slici 4.8.

Ako je korisnik slučajno odabrao neku vrstu piva i spremio tada ju može obrisati iz liste tako da pritisne njeno ime. Listu prethodno odabranih piva korisnik može otvoriti i tako da u navigacijskoj komponenti na dnu zaslona odabere zadnju ikonu *OTHER* te kada se otvori taj zaslon pritisne dugme *PRESETS* kao što je vidljivo na slici 4.9.



Slika 4.8. *Odabrane vrste pive.*

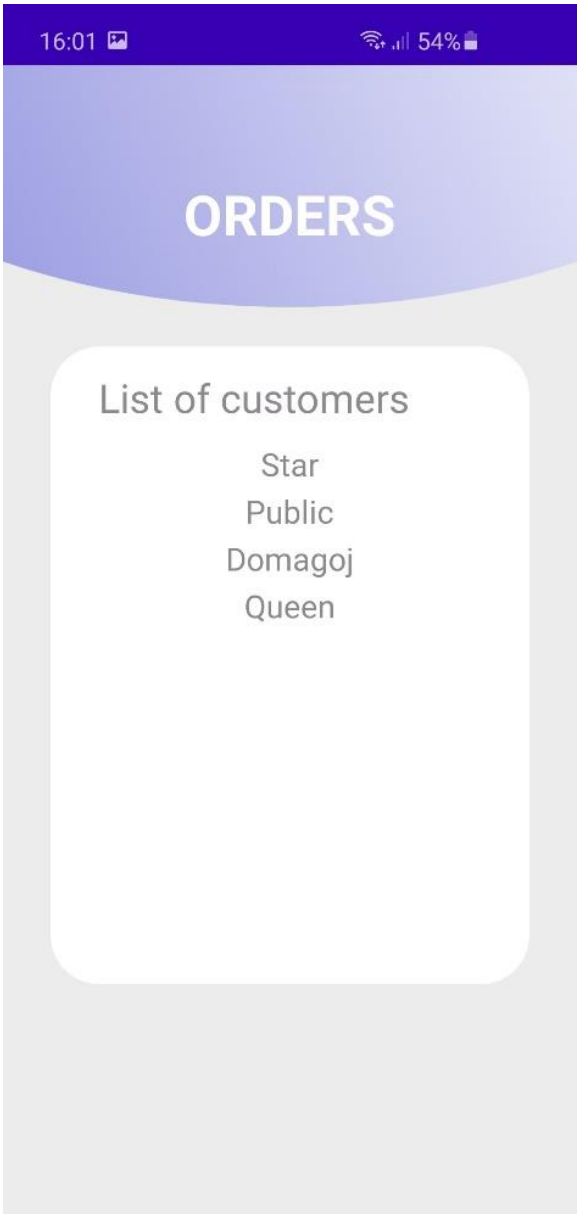
Na *OTHER* zaslonu se nalaze još dugmad za odjavljivanje prijavljenog korisnika *LOGOUT* i dugme za prikaz svih narudžbi klijenata *ORDERS*.



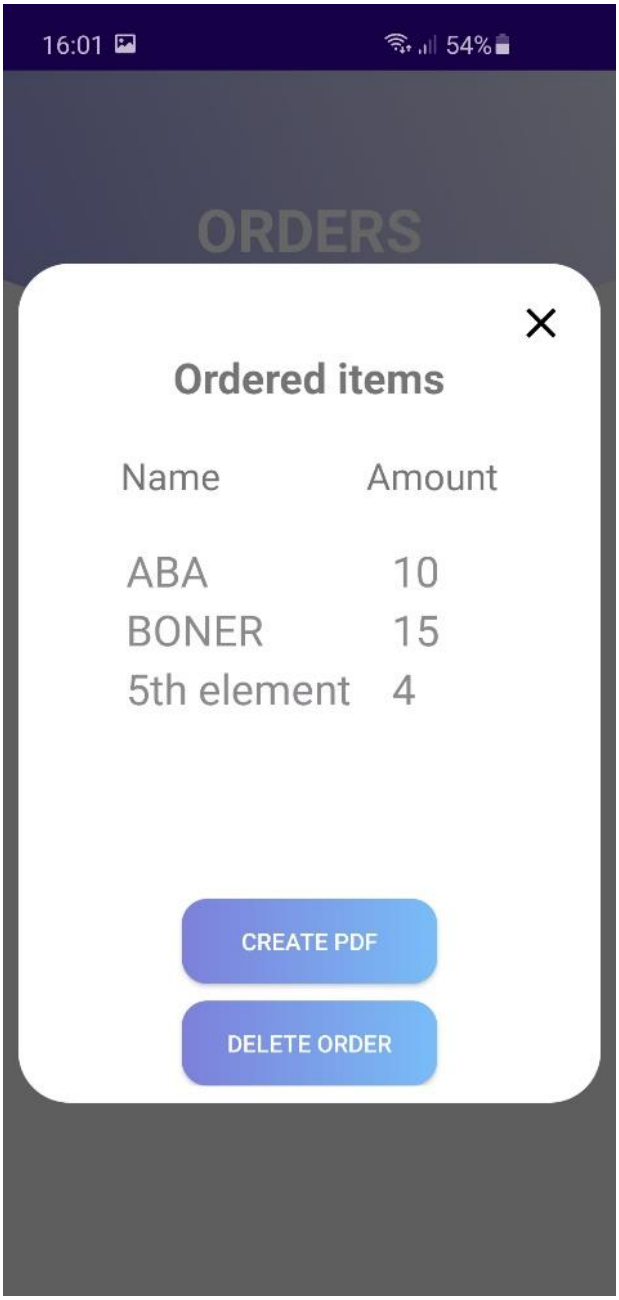
Slika 4.9. *Other zaslon.*

4.4 Kreiranje i spremanje narudžbe klijenta

Pritiskom dugmeta *ORDERS* korisniku se otvara zaslon za prikaz svih narudžbi prikazan na slici 4.10. Svaku narudžbu se može odabrati i spremiti u pdf format ili obrisati ako je korisnik krivo unio podatke u narudžbu. Na slici 4.11. može se vidjeti dijalog odabrane narudžbe.

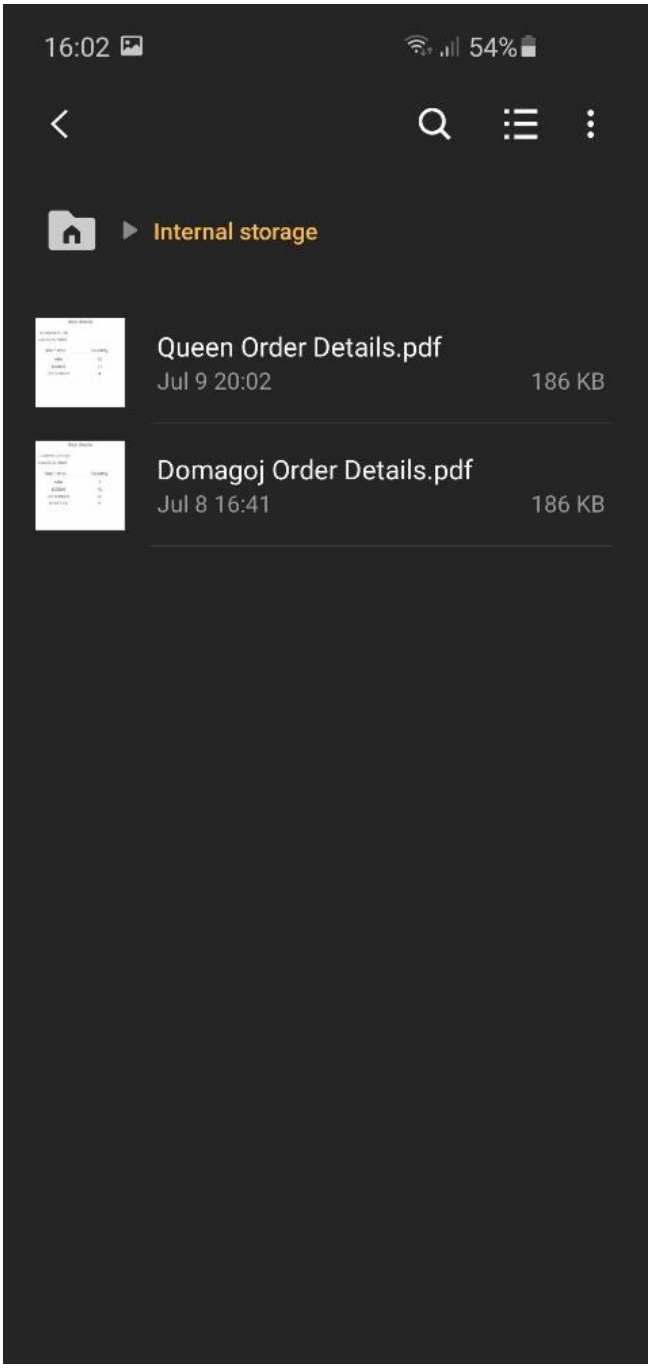


Slika 4.410. *Zaslon s narudžbama.*



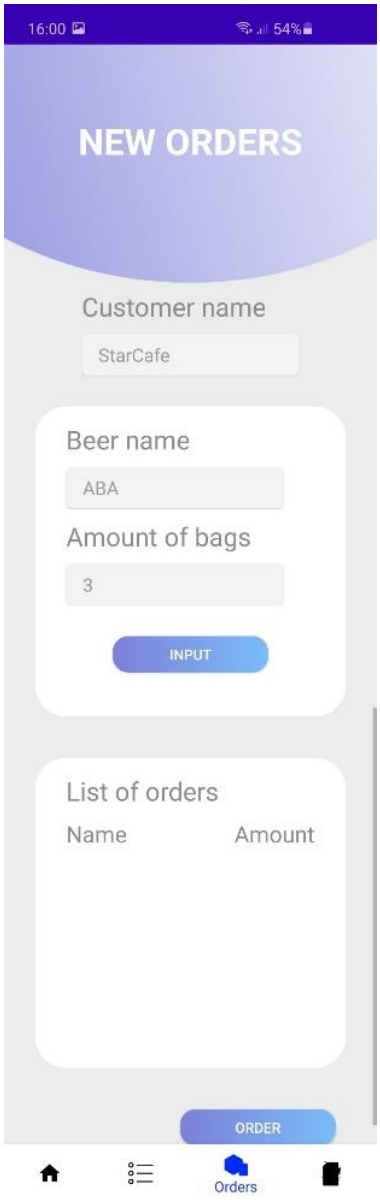
Slika 4.411. *Dijalog odabrane narudžbe.*

Odabirom dugmeta *CREATE PDF* korisnik sprema narudžbu u pdf format u unutarnju pohranu uređaja, a odabirom dugmeta *DELETE ORDER* narudžba se briše. Prikaz primjera unutarnje pohrane podataka na slici 4.12. Sada kad je narudžba spremljena u unutarnju pohranu uređaja korisnik može manipulirati s njom kako želi, na primjer poslati nadređenom za spremanje i slanje narudžbe.



Slika 4.4.12. *Prikaz unutarnje pohrane narudžbe na uređaju.*

Do sada je prikazan način prikaza kreiranih narudžbi. Kako bi se one kreirale korisnik mora otići u *ORDERS* zaslon. U navigacijskoj komponenti korisnik odabire ikonu nalik na kutije, te mu se otvara zaslon *ORDERS*. Na slici 4.13. je prikaz zaslona za kreiranje nove narudžbe.



Slika 4.413. *Zaslon za kreiranje nove narudžbe.*

Prilikom uzimanja i kreiranja nove narudžbe klijenta aplikacija sakriva unos vrsta piva sve dok korisnik ne unese ime klijenta. Nakon što je uneseno ime klijenta tek tada se mogu unositi vrste piva u listu koje klijent želi naručiti. Pive se naručuju tako što korisnik unese ime piva i količinu piva u gajbama, te pritiskom na dugme *INPUT* unosi tu pivu u listu koja se sastoji od svih piva koje je korisnik unio. Obavljenim naručivanjem korisnik pritišće dugme *ORDER* koje uzima narudžbu i sprema u *ORDERS* listu na zaslonu odabranim narudžbama koja je vidljiva na slici 4.13.

5. ZAKLJUČAK

Cilj ovog rada je bio olakšati i unaprijediti u digitalnom smislu upravljanje skladištem pivovare. Kako danas skoro svi koriste mobilne uređaje i imaju pristup uređajima konstantno, ideja je bila da korisnik kako prolazi skladištem unosi podatke koji su relevantni. Aplikacija je radi čitljivosti i sigurnosti napravljeno što jednostavnijeg izgleda, zahtijeva povezanost na internet radi baze podataka i korisnikovo dopuštenje za korištenje lokalnog spremanja podataka radi spremanja pdf-a. Korisnik koji nije registriran, registrira se i aplikacija ga odmah odvodi do pogleda na stanje skladište. Unos novih sastojaka je promišljen i fokusiran na ljudsku pogrešku kako se ne bi dogodilo da ima više istoimenih sastojaka ili unošenje prekomjerne količine. Također su implementirane funkcije kreiranja jednostavnih narudžbi koje se spremaju u pdf dokument i kreiranje predefiniranih vrsta piva koje se proizvode. Tako se svi sastojci iz skladišta oduzmu od piva koja se proizvodi i tako daje uvid u preostale količine u skladištu.

Aplikacije pokriva sve osnovne segmente koje zahtijeva vođenje skladišta. Od mogućih nadogradnji i poboljšanja mogu biti postavljanje datuma i vremena odabranog predefiniranog piva i u listi prikaza odabranih vrsta i kreiranih narudžbi. Od poboljšanja moguće je napisati aplikacija u Kotlin jeziku i u arhitekturi kao što su MVC, MVVM i MVP.

LITERATURA

[1] Povijest Android mobilnog operativnog sustava, https://www.androidauthority.com/history-android-os-name-789433/, pristupljeno: srpanj, 2020.

[2] Android studio, https://developer.android.com/studio/intro/?gclid=Cj0KCQjwvIT5BRCqARIsAAwwD-SQl5I9-kpnfVk19AOskZwFM6PckrrUrUPZ8_YqhZxX30R7A6jopxgaAujMEALw_wcB&gclsrc=aw.ds, pristupljeno: srpanj, 2020.

[3] Java, https://hackr.io/blog/what-is-java, pristupljeno: srpanj, 2020.

[4] Adobe XD, https://designshack.net/articles/software/what-is-adobe-xd/, pristupljeno: srpanj, 2020.

[5] Firebase, https://medium.com/firebase-developers/what-is-firebase-the-complete-story-abridged-bcc730c5f2c0, pristupljeno: srpanj, 2020.

[6] Inačice Androida, https://www.androidauthority.com/history-android-os-name-789433/, pristupljeno: srpanj, 2020.

[7] Git, https://www.atlassian.com/git/tutorials/what-is-git, pristupljeno: srpanj, 2020.

SAŽETAK

Naslov: Android aplikacija za vođenje skladišta pivovare

Kako bi se digitaliziralo poslovanja pivovare, konkretno vođenje skladišta kreirana je *Android* aplikacija. Aplikacija je ponajviše kreirana kako bi se izbjegla ljudska pogreška prilikom izračuna i unosa. Prije nego li je aplikacija izrađena skladištar bi morao ručno unositi na papir sve sastojke koji bi dolazili u pivovaru i tako se ta informacija mogla vrlo lako zagubiti. S aplikacijom korisnik bi se prijavio i dobio bi prikaz stanja u skladištu svoje pivovare koji je osmišljen da bude što jednostavniji i čitljiviji. Kada bi došla nova pošiljka sastojaka u skladište korisnik bi unio količinu te bi aplikacija sama odradila statistički dio kao što je količina pojedinih sastojaka u skladištu, te ostatak slobodnog mjesta u skladištu. Svi podaci o skladištu su povezani s online bazom podataka koje se ažurira u realnom vremenu kako bi korisnik imao trenutan prikaz onoga što se događa. Uz vođenje skladišta korisnik može kreirati nove vrste piva koje se planiraju proizvoditi i primiti narudžbu klijenta. Kada se kreće u proizvodnju piva, odabire se pivo koje se kreće proizvoditi i aplikacija bi oduzela sve sastojke potrebne za tu vrstu piva, te spremila naziv tog piva u listu piva u proizvodnji. Kod kreiranja narudžbi korisnik unosi naziv klijenta, vrste piva koje želi naručiti, te količinom tog piva. Nakon izrade narudžbe korisnik može spremiti narudžbu u dokument i poslati nadređenima za slanje narudžbe u pivovari.

Ključne riječi: Android, Firebase, skaldište, pivo, upravljanje

ABSTRACT

Title: Android application for managing beer warehouse

In order to digitize the business of the brewery, specifically warehouse management Android application was created. The application is mostly created to avoid human error when calculating and entering new information. Before the application was made, the storekeeper would have to manually enter on paper all the ingredients that would come into the brewery and so that information could be very easily lost. With the app, the user would log in and get a view of the status of their brewery's warehouse, which is designed to be as simple and readable as possible. When a new shipment of ingredients arrived in the warehouse, the user would enter the quantity and the application would do the statistical part itself, such as the quantity of individual ingredients in the warehouse, and the rest of the free space in the warehouse. All warehouse data is connected to an online database that is updated in real time so that the user has an instant view of what is happening. With warehouse management, the user can create new types of beer that are planned to be produced and receive a customer order. When they would start brewing beer, they would select the beer to be produced and the application would take away all the ingredients needed for that type of beer, and save the name of that beer in the list of beers in production. When creating orders, the user enters the name of the client, the type of beer he wants to order, and the quantity of that beer. After creating the order, the user can save the order in a document and send to superiors for sending orders out to a clients.

Keywords: Android, beer, Firebase, managing, warehouse

ŽIVOTOPIS

Valerian Bahnik, rođen 27.2.1998. godine u Virovitici, nakon završene osnovne škole Vladimira Nazora u Daruvaru upisuje srednju strukovnu školu u Daruvaru smjer tehničar za računarstvo. Nakon završetka Tehničke škole u Daruvaru 2016. godine upisuje preddiplomski stručni studij elektrotehnike, smjer informatike na Fakultetu elektrotehnike, računalstva i informacijskih tehnologija u Osijeku. Slobodno vrijeme provodi istražujući nove tehnologije i osobnim treniranjem. Obveznu praksu na trećoj godini studija dobiva i obavlja u firmi Mono usavršavajući se o načinu rada u timovima i izradi većih projekata. Izradom projekta ući važnost kvalitetne arhitekture kao što je *Onion*, te izrada čistog koda. Neke od tehnologija koje koristi tijekom izrade *Android studio, Adobe XD, Visual Studio Code, Source tree* i *Slack.*

PRILOZI

Projektna mapa s izvornim kôdom i priloženim word i pdf dokumentom nalazi se na optičkom disku koji je priložen uz printanu verziju završnog rada.