Android aplikacija za upravljanje narudžbama u restoranima

Majstorović, Luka

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

Sveučilišni studij

APLIKACIJA ZA UPRAVLJANJE NARUDŽBAMA U RESTORANIMA

Završni rad

Luka Majstorović

Osijek, 2023.

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

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

Osijek, 14.09.2023.

Odboru za završne i diplomske ispite

Prijedlog ocjene završnog rada na preddiplomskom sveučilišnom studiju

Ime i prezime Pristupnika:	Luka Majstorović
Studij, smjer:	Programsko inženjerstvo
Mat. br. Pristupnika, godina upisa:	R4534, 27.07.2020.
OIB Pristupnika:	94360120406
Mentor:	izv. prof. dr. sc. Ivica Lukić
Sumentor:	,
Sumentor iz tvrtke:	
Naslov završnog rada:	Android aplikacija za upravljanje narudžbama u restoranima
Znanstvena grana rada:	**Informacijski sustavi (zn. polje računarstvo)**
Zadatak završnog rad:	U sklopu završnog rada potrebno je kreirati android aplikaciju za upravljanje narudžbama u restoranima s ciljem ubrzanja i pojednostavljenja cijelog procesa. Aplikacija podržava dva načina rada. Prvi način rada je administrator, koji prijavljuje restoran, također dodaje, uređuje i briše osoblje i stavke u jelovniku. Drugi način rada je osoblje, koje upisuje narudžbe i označuje ih kao dovršene. Sustav bilježi osoblje koje je rukovalo narudžbom. Tema je rezervirana za: Luka Majstorović
Prijedlog ocjene završnog rada:	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 2 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 2 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene od strane mentora:	14.09.2023.
Datum potvrde ocjene od strane Odbora:	24.09.2023.

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

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

IZJAVA O ORIGINALNOSTI RADA

Osijek, 24.09.2023.

Ime i prezime studenta:	Luka Majstorović
Studij:	Programsko inženjerstvo
Mat. br. studenta, godina upisa:	R4534, 27.07.2020.
Turnitin podudaranje [%]:	2

Ovom izjavom izjavljujem da je rad pod nazivom**: Android aplikacija za upravljanje narudžbama u restoranima**

izrađen pod vodstvom mentora izv. prof. dr. sc. Ivica Lukić

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:

Majstorović

SADRŽAJ

1. UVOD

Kada je neka osoba gost u ugostiteljskoj instituciji postoji par stvari na koju osoba obraća pozornost. Jedna od važnih stavaka u iskustvu gosta jest način na koji se odvija proces naručivanja stavki iz ponude u ugostiteljskoj instituciji. Vrlo je važno da taj proces bude brz, da se ne događaju greške te da se moguće požaliti na uslugu. Kako bi se proces učinio kontroliranim i standardiziranim, postoji potreba za nekim sustavom rukovanja narudžbama. Postoji nekoliko mogućnosti koje se često koriste u ugostiteljskoj industriji. Prvi način jest da poslužitelj upamti narudžbu napamet, što često rezultira time da netočne informacije dođu do osoblja koje priprema narudžbe, ili se izostavi stavka sa narudžbe. Uz to postoji nepotrebni hod do dijela objekta gdje se pripremaju narudžbe. Drugi način jest zapisivanje na papir, što eliminira zaboravnost poslužitelja, no još uvijek postoji nepotrebni hod i proces zapisivanja je spor. U ovom digitalnom vremenu bilo bi dobro naći rješenje koje eliminira sve negativne posljedice rukovanja narudžbama na raznorazne načine. Rješenje bi bilo napraviti aplikaciju koja omogućuje automatsku komunikaciju između osoblja koje poslužuje goste te osoblja koje priprema narudžbe. Uz to, aplikacija bi imala sučelje koje omogućuje brzo zapisivanje stavki u narudžbu, uvelike olakšavajući posao osoblju. Narudžbe bi sve bile zapisane i čuvane u nekakvoj bazi podataka te bi uvijek bile dostupne u slučaju eventualne žalbe gosta. Uz ove pogodnosti, postoje još mnoge moguće pogodnosti koje se mogu dodati daljnjim razvijanjem sustava, tako da sustav nije ograničen, već se može prilagoditi potrebama korisnika.

1.1. Zadatak završnog rada

U sklopu završnog rada potrebno je kreirati Android aplikaciju za upravljanje narudžbama u restoranima s ciljem ubrzanja i pojednostavljenja cijelog procesa. Aplikacija podržava dva načina rada. Prvi način rada je administrator, koji prijavljuje restoran, također dodaje, uređuje i briše osoblje i stavke u jelovniku. Drugi način rada je osoblje, koje upisuje narudžbe i označuje ih kao dovršene. Sustav bilježi osoblje koje je rukovalo narudžbom.

2. PREGLED POSTOJEĆIH RJEŠENJA

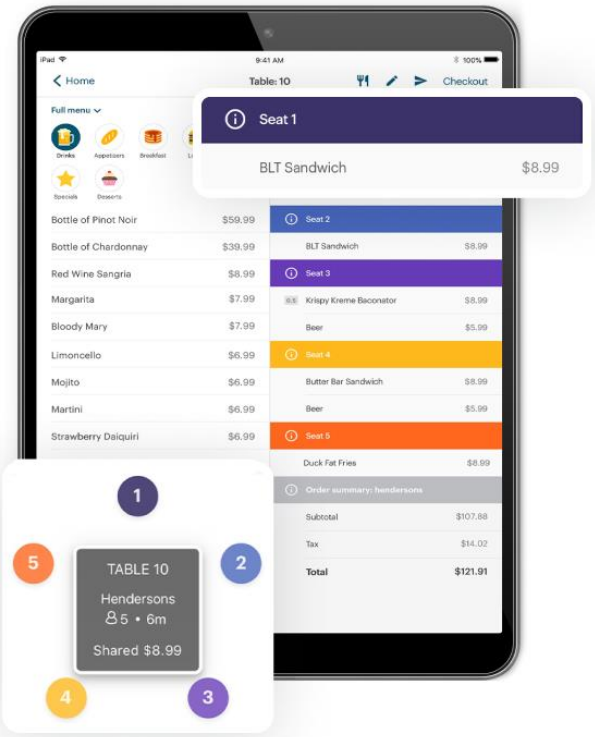
Radi boljeg razumijevanja zadatka potrebno je napraviti analizu već implementiranih rješenja koja se koriste u industriji. Tom analizom moguće je uvidjeti karakteristike dizajna, sličnosti među raznim rješenjima te neke jedinstvene implementacije određenih značajki programa.

2.1. Orderman

Slika 2.1. Prikaz Orderman sustava [1]

Orderman (Slika 2.1) je rješenje popularno na području Europe koje je implementirano na POS uređajima. Glavna funkcionalnost je omogućavanje upravljanja narudžbama na način da osoblje koje uslužuje i osoblje koje priprema narudžbe komuniciraju putem POS uređaja. Osoblje koristi posebne mobilne uređaje kako bi unosili narudžbe u sustav te tada drugi dio osoblja ima uvid u aktivne narudžbe. Orderman nudi i nosivi pisač za račune koji omogućuje ispisivanje računa direktno pred gostima. Uz to omogućuje i razdvajanje računa na više dijelova kako bi svaka osoba bila u mogućnosti dobiti račun isključivo za svoju narudžbu [2].

2.2. TouchBistro



Slika 2.2. Prikaz TouchBistro sustava [3]

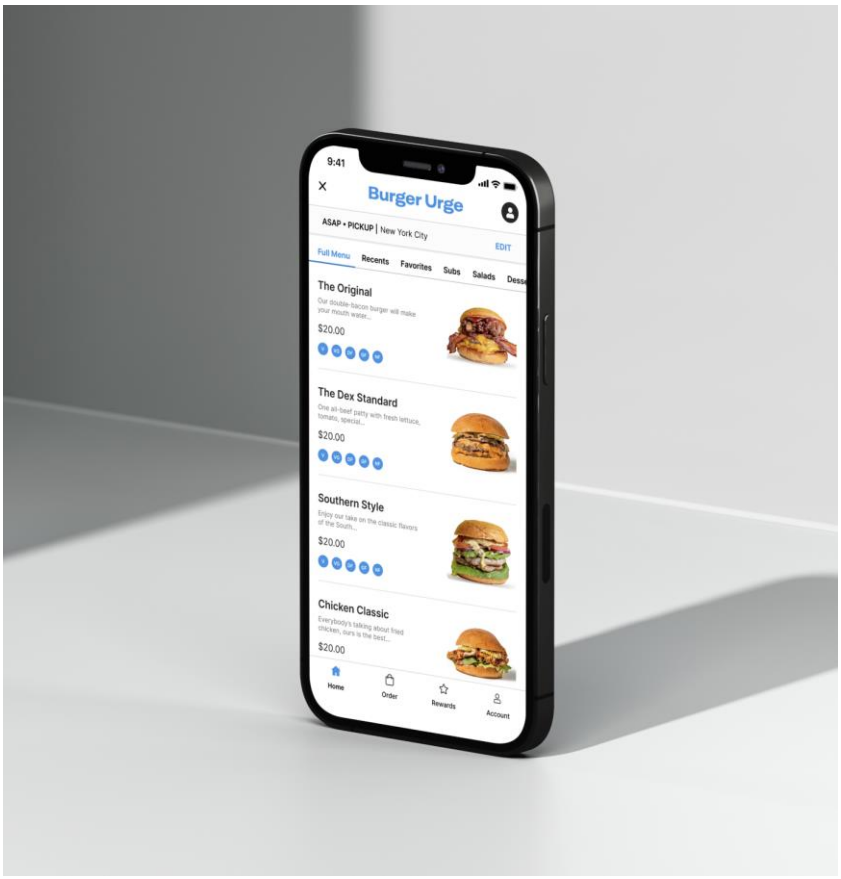
TouchBistro (Slika 2.2) temelji svoju implementaciju na POS aplikaciji na iPad-u. Pošto nema potrebe za specijaliziranim uređajima, ovakva implementacija je u startu pristupačnija svojom fleksibilnošću bez gubitka svojih glavnih funkcionalnosti. Aplikacija nudi cjelokupnu funkcionalnost upravljanja narudžbama, mogućnost unošenja rasporeda smjena osoblja, upravljanje zalihama i dr. Narudžbe se mogu unositi pored stola, a poslovođe imaju pristup pregledu prodajnih aktivnosti [4].

2.3. Revel Systems

Slika 2.3. Prikaz Revel Systems sustava [5]

Revel Systems ističe se kao još jedno rješenje koje implementira POS platformu na iPad-u. Komunikacija među uređajima odvija se putem digitalnog oblaka. Funkcionalnosti koje ova aplikacija nudi su rukovanje transakcijama, upravljanje zalihama, upravljanje zaposlenicima, itd. Aplikacija ima mogućnost plaćanja čak i u privremenoj nemogućnosti spajanja na internet mrežu. U trenutku kada ponovno dobije pristup internetskoj mreži automatski obrađuje plaćene račune. Sustav je osmišljen na način da aplikacija ima tok rada jednak onom uobičajenom toku razmišljanja gosta što omogućuje tečan proces upisivanja narudžbe u aplikaciju.

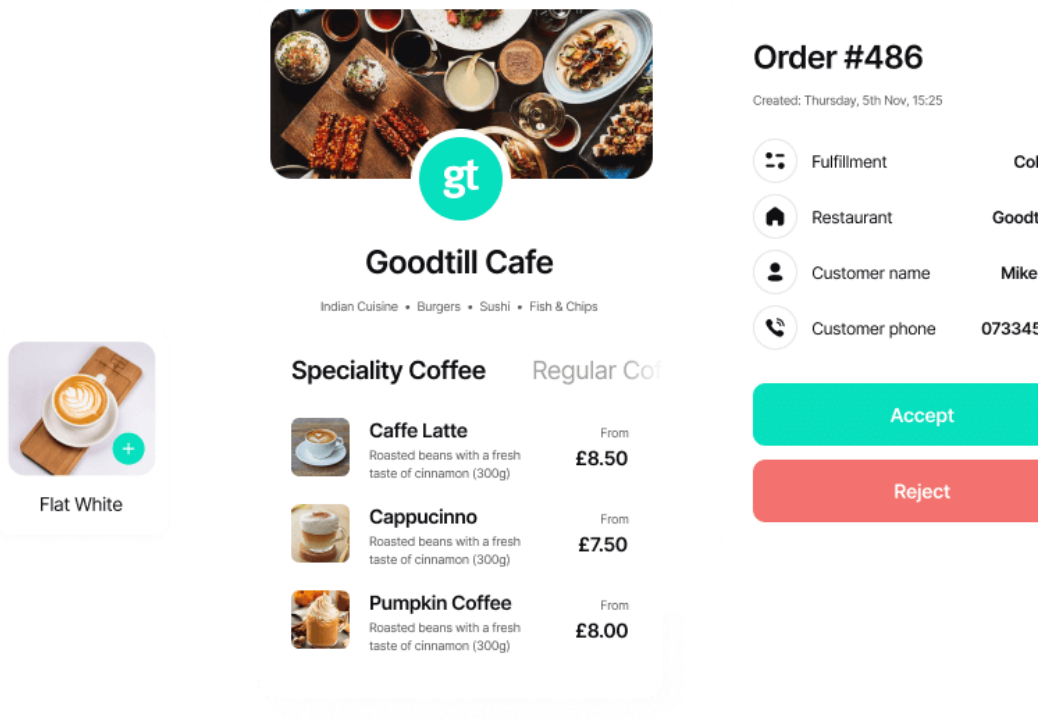
2.4. Lunchbox



Slika 2.4. Prikaz Lunchbox sustava [6]

Lunchbox je alat za upravljanje restoranima s mogućnostima naručivanja preko mreže, upravljanja odnosa s kupcima (engl. *customer relationship management*), agregacija naloga (engl. *order agregation*) i upravljanja narudžbama uz stol. Ovaj alat ujedno koriste i restorani i gosti restorana. Gosti imaju uvid u digitalne jelovnike restorana i postoji mogućnost plaćanja Apple Pay ili Google Pay sustavima. Platforma je ujedno i namijenjena da radi sa većinom POS uređaja tako ju je lako uvesti kao novu tehnologiju u restoran.

2.5. Goodeats



Slika 2.5. Prikaz Goodeats sustava [7]

Goodeats je namijenjen biti klikni-i-pokupi (engl. *click-and-collect*) sustav i sustav koji omogućava rukovanje naručivanjem pored stola. Platforma se postavlja jednostavno, što je vrlo važna stavka ovakvog tipa sustava. Još neke od značajki su procesuiranje plaćanja (engl. *payment processing*) i stvaranje jedinstvenog URL do određene stranice za naručivanje. Goodeats uključuje i POS sustav, upravljanje dionicama, izvješćivanje i analitiku, sustav kuhinjskog zaslona i sustav lojalnosti (engl. *loyalty system*).

3. PREGLED KORIŠTENIH TEHNOLOGIJA

Kako bi bilo moguće osmisliti dizajn same aplikacije, prvo je potrebno promisliti o tehnologijama koje bi bile potrebne za razvoj aplikacije. U nastavku su opisane glavne tehnologije koje su korištene za izradu različitih segmenata aplikacije.

3.1. Kotlin

Kotlin je programski jezik otvorenog koda (engl. *open-source*) sa statičnim (engl. *static*) tipovima podataka. Podržava objektno orijentiranu, kao i funkcionalnu paradigmu. Sintaksa samog jezika slična je programskim jezicima poput Jave, C#, Scala i drugih. Službeno ga podržava Google za izradu Android aplikacija i zbog toga, za nove aplikacije, Kotlin je postao službeni standard. Zbog velikog broja aplikacija koje su napravljene u Javi, kako bi se potakao prelazak na Kotlin, omogućeno je međusobno pozivanje funkcija napisanih u Javi i Kotlinu, tako da se postupno može ažurirati već napisani kod u Javi, bez većih pauza u stvaranju novih komponenata. Na taj način Kotlin se uspješno probio uz velike potpore Google-a i danas je standard za Android aplikacije [8].

3.2. Jetpack Compose

Jetpack Compose je deklarativni alat za stvaranje korisničkog sučelja u Android aplikacijama. Sustav koji se upotrebljavao prije Jetpack Compose-a koristio je XML za slaganje korisničkog sučelja, no Compose koristi intuitivan način koji zahtjeva manje linija koda, ima snažne alate i API implementacije koje pružaju već gotovu logiku, poput Material Design API implementacije. Compose je u skladu i prati trendove najnovijih tehnologija drugih tržišta poput SwiftUI razvojnog okvira (engl. *framework*) tvrtke Apple [9].

3.3. Android Studio

Android Studio je službeno integrirano razvojno okruženje (engl. *integrated development environment*) za stvaranje Android aplikacija. Koristi uređivač koda (engl. *code editor*) kojeg održava IntelliJ IDEA. U razvojnom okruženju moguće je stvoriti projekte iz nule s raznim opcijama, što omogućava brzo postavljanje programa s osnovnim funkcionalnostima i standardiziranom strukturom projekta. Sadrži i emulator za simuliranje različitih Android uređaja, koristi Gradle sustav izgradnje i daje još mnogo mogućnosti. Za pisanje gotovo svog

koda i testiranje aplikacije, dovoljno je imati ovo razvojno okruženje jer je vrlo snažan i raznolik alat [10].

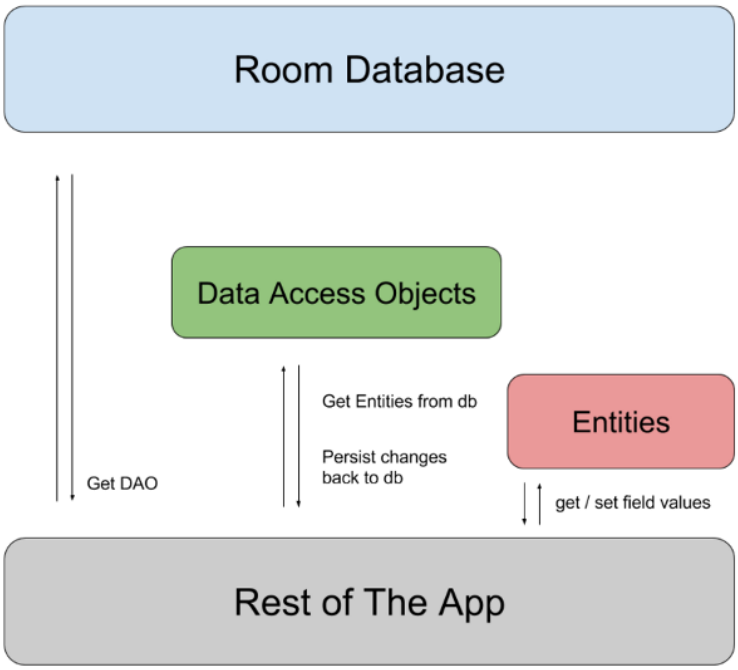
3.4. Baze podataka

Unutar ove aplikacije postoji potreba za dvjema vrstama baza podataka, a to su mrežna baza podataka i lokalna baza podataka. Kako se ne bi opteretio rad mrežne baze podataka, u lokalnu bazu podataka se sprema određeni dio podataka koji se ne dijeli preko više uređaja.

3.4.1. Firebase – Cloud Firestore

Cloud Firestore, alat Firebase-a i Google Cloud-a služi kao baza podataka koja je jednostavna za postavljanje, fleksibilna te podržava rad u stvarnom vremenu. Omogućava postavljanje slušatelja na njezine zbirke podataka te ih obavještava o promjenama unutar zbirki, što programu olakšava otkrivanje promjena u sustavu. Najistaknutija primjena Cloud Firestora u ovoj aplikaciji jest spremanje i dohvaćanje podataka unutar baze putem internetske mreže. Time se omogućava komunikacija više uređaja u stvarnom vremenu. Stoga se ova baza podataka koristi za podatke koji moraju biti dostupni većem krugu uređaja, a ne samo trenutnom korisniku [11].

3.4.2. Room baza podataka



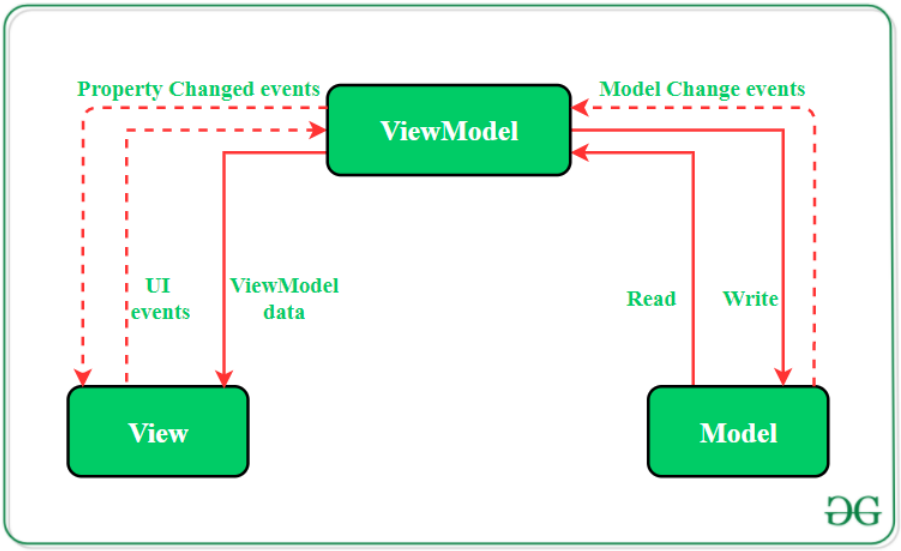
Slika 3.1. Prikaz dijagrama arhitekture Room baze podataka [12]

Room baza podataka je lokalna baza podataka u sklopu Android Jetpack biblioteke. Apstrahira (engl. *abstracts*) i daje jednostavno sučelje za rad s lokalnom SQL bazom podataka. U Room se spremaju podatci koji se nemaju potrebe dijeliti s drugim uređajima, odnosno podatci koji su potrebni isključivo za rad trenutnog uređaja [13].

3.5. Koin

Koin je razvojni okvir za ubrizgavanje ovisnosti za Kotlin programski jezik. Pomoću Koin-a, moguće je na jednostavan način stvarati jednočlane (engl. *singleton*) objekte, ubrizgavati ovisnosti u te objekte te općenito imati skalabilnu (engl. *scalable*) strukturu aplikacije, tako da bude mnogo lakše nadograditi aplikaciju novim ovisnostima na jednostavan način nakon prvotne definicije strukture sustava. Koin se u ovoj aplikaciji upotrebljava putem svojih modula [14].

3.6. MVVM arhitektura



Slika 3.2. Prikaz MVVM arhitekture [15]

Model-Prikaz-ModelPrikaza (engl. *Model-View-ViewModel)* je arhitekturni obrazac koji odvaja prezentacijsku logiku od poslovne logike android aplikacije. Sadrži se od tri glavna koncepta: model, prikaz i model prikaza. Model skriva rukovanje podatcima koje odrađuje podatkovni sloj. Prikaz je odgovoran samo za prikazivanje podataka te obavještavanje modela prikaza o korisničkim radnjama, no ne smije sadržavati nikakvu logiku u sebi. Na posljetku, model prikaza izlaže tokove podataka (engl. *data flows*) koji su potrebni prikazu i rukuje s tim podatcima te

služi kao poveznica između modela i prikaza. Slika 3.2. prikazuje dijagram koji opisuje arhitekturu i veze među komponentama [16].

4. DIZAJNIRANJE SUSTAVA

U ovom dijelu bit će objašnjen dizajn samoga sustava, odnosno glavne ideje koje su vodile razvoj aplikacije. Ovaj korak je izrazito važan jer se ovdje definira smjer u kojem ide razvoj aplikacije. Važan je i jer naknadne promjene u dizajnu sustava rezultiraju obujmom posla daleko većim nego onim kada bi se dizajn točno definirao prije početka implementacije. Često se ističe kao korak kojega rade osobe s mnogo iskustva zbog svoje kompleksnosti te utjecaja koji ima na daljnji razvoj. Iako loš dizajn možda nije vidljiv u trenutku kada se dizajn stvara, kada dođe do implementacije sustava, često se otkriju rupe i mane u osmišljenom dizajnu i tada mijenjanje dizajna stvara mnogo više posla bespotrebno.

Dizajn ove aplikacije se ugrubo može podijeliti u tri dijela:

a. Prvi dio bi bio funkcionalni ili konceptualni dizajn. Funkcionalni dizajn definira što aplikacija treba ostvariti, koja je njena svrha i koje značajke treba sadržavati. Ne bavi se pitanjem kako implementirati te zahtjeve, već te same zahtjeve zadaje. Piše se u jeziku koji klijent razumije bez tehničkog znanja.
b. Drugi dio bi bio tehnički dizajn. Tehnički dizajn se za razliku od funkcionalnog dizajna bavi pitanjem načina implementacije zahtjeva zadanih u funkcionalnom dizajnu. U sklopu stvaranja ovog dizajna odlučuje se o tehnologijama koje će se koristiti, način na koji će se određene značajke implementirati, arhitektura sustava, načela po kojima će se sustav razvijati, programski jezici, itd.
c. Posljednji dizajn bi bio dizajn korisničkog sučelja. Dizajn korisničkog sučelja nije nužno zadnji korak dizajniranja niti onaj najmanje važan. Kod dizajna korisničkog sučelja, stvara se izgled i korisničko iskustvo same aplikacije i to je inače dio koji može biti razlog uspjeha ili neuspjeha aplikacije na tržištu.

Unutar ovog poglavlja opisan je funkcionalni i tehnički dizajn aplikacije.

4.1. Funkcionalni dizajn

Svrha ove aplikacije jest omogućiti restoranima jedno sučelje za upravljanje narudžbama. Iako na prvu ovaj zahtjev izgleda jednostavno, potrebno ga je razraditi. Potrebno je postaviti pitanja poput „Tko koristi ovu aplikaciju?“, „Koje informacije su potrebne u jednoj narudžbi?“ ili pak „Zašto je uopće potrebno ovakvo rješenje?“. Sa jednim pitanjem, otvara se još više pitanja i

proces stvaranja funkcionalnog dizajna ne staje sve dok sva pitanja koja se tiču zahtjeva nisu odgovorena.

Aplikacija je osmišljena za dva tipa korisnika: konobar i administrator. Svaki korisnik ima podatke o svome imenu i prezimenu, odabranom korisničkom imenu i lozinki. Sadrži i identifikacijsku oznaku ugostiteljskog objekta za koji se korisnik registrira, vrijeme registracije korisnika, ulogu koja se pri registraciji automatski dodjeljuje na ulogu konobara te informaciju o tome je li korisnički račun odobrio administrator ugostiteljskog objekta. Korisničko ime mora biti jedinstveno, a lozinka mora sadržavati barem 8 znakova od kojih postoji barem jedan znak iz svake od kategorija: malo slovo, veliko slovo, brojka i simbol.

Konobar može stvarati nove narudžbe, označiti narudžbe završenima ili otkazati narudžbe, uz to ima mogućnost vidjeti informacije o dovršenim narudžbama u slučaju upita ili nekakvog nesporazuma. Narudžba treba sadržavati informacije o naručenim stavkama i njihovim količinama te o stolu na koji je naručena kako bi osoblje znalo gdje narudžba treba biti odnesena nakon što je pripremljena. Nadalje, narudžba sadržava svoje vrijeme stvaranja radi lakšeg snalaženja pri pronalasku narudžbi u povijesti narudžbi. Isto tako, narudžba sadrži identifikacijske oznake osoblja koje je bilo zaslužno za stvaranje i dovršavanje narudžbe. Pri stvaranju narudžbe, konobar mora moći dodati ili ukloniti, odnosno povećati ili smanjiti količinu određenih stavki unutar narudžbe. Mora moći unijeti broj stola za koji se narudžba stvara. U svakom trenutku mora se moći odjaviti i vidjeti svoje ime dok je prijavljen u aplikaciju. Ujedno mora imati sustav navigacije između različitih ekrana. Korisniku konobar omogućeno je registriranje u određenom ugostiteljskom objektu. Pri registraciji unosi svoje podatke.

Drugi korisnik, odnosno administrator, može upravljati korisnicima, dodavati i uklanjati stavke iz jelovnika i može upravljati aktivnim i dovršenim narudžbama. Korisnicima upravlja na način da ima odvojen prikaz odobrenih i neodobrenih korisničkih računa te ima pregled svih podataka i može ih mijenjati isključujući datum stvaranja korisničkog računa. Nije dozvoljeno mijenjati korisničko ime ukoliko takvo već postoji, isto tako ukoliko se lozinka mijenja, mora pratiti pravila kao i pri registraciji, odnosno mora sadržavati barem 8 znakova od kojih postoji barem jedan znak iz svake od kategorija: malo slovo, veliko slovo, brojka i simbol. Kod upravljanja jelovnikom, administrator može dodavati nove stavke za jelovnik te brisati postojeće. Pregled i upravljanje postojećim narudžbama ili onim već dovršenima je jednako kao i kod konobara. Administrator nema mogućnost stvaranja novih narudžbi.

Podatci o narudžbama, korisnicima i jelovniku moraju biti dostupni korisniku koji im pristupa u stvarnom vremenu. Promjene podataka se moraju odražavati u kratkom vremenskom roku na svim uređajima koji pristupaju podatcima u vrijeme njihove promjene. To će osiguravati upućenost svih uključenih korisnika u trenutno stanje.

4.2. Tehnički dizajn

4.2.1. Podatkovni sloj

Ključan dio podatkovnog sloja su modeli. Modeli su raspodijeljeni u dvije baze podataka. Jedna baza podataka jest lokalna baza podataka implementirana putem Room baze podataka. U nju se spremaju stavke narudžbe koja nije potvrđena, odnosno treba biti dostupna samo lokalno, kako bi korisnik mogao uređivati narudžbu prije nego ju upotpunjenu pošalje u bazu dostupnu svim korisnicima.

Model stavke u narudžbi koja nije potvrđena se sastoji od sljedećih podataka: identifikacijska oznaka te količina tipa cijeli broj (engl. *integer*), uz to i naziv tipa niz znakova (engl. *string*).

Druga baza podataka jest mrežna baza podataka implementirana putem Firebase Firestore. Firestore omogućava komunikaciju u stvarnom vremenu i dijeljenje podataka među uređajima. Stoga oni podatci koji su ključni za rad više korisnika moraju se skladištiti u mrežnu bazu podataka.

Model korisnika se sastoji od sljedećih podataka: identifikacijska oznaka, korisničko ime, lozinka, ime i prezime, uloga te identifikacijska oznaka ugostiteljskog objekta tipa niz znakova. Uz to, sadrži i podatak o tome je li korisnik odobren tipa Booleove vrijednosti (engl. *boolean*) te vrijeme stvaranja tipa vremenska oznaka (engl. *timestamp*).

Model narudžbe se sastoji od sljedećih podataka: identifikacijska oznaka, identifikacijska oznaka ugostiteljskog objekta, broj stola, identifikacijska oznaka korisnika koji stvara narudžbu te identifikacijska oznaka korisnika koji dovršava narudžbu tipa niz znakova, uz to i podatak o tome je li narudžba aktivna tipa Booleove vrijednosti te vrijeme stvaranja tipa vremenska oznaka.

Model stavke u narudžbi se sastoji od sljedećih podataka: identifikacijska oznaka, identifikacijska oznaka narudžbe te naziv tipa niz znakova, uz to i podatak količini tipa cijeli broj.

Model stavke u jelovniku se sastoji od sljedećih podataka: identifikacijska oznaka, naziv te identifikacijska oznaka ugostiteljskog objekta tipa niz znakova.

Model ugostiteljskog objekta se sastoji od sljedećih podataka: identifikacijska oznaka te naziv tipa niz znakova.

Svaki od ovih modela ima svoj model za bazu podataka te model za aplikaciju koji se previše ne razlikuju jedan od drugog, ali pružaju razdvajanje u odvojene jedinice s ciljem smanjivanja preklapanja među funkcionalnostima pojedinih jedinica i skrivanja implementacije među slojevima aplikacije.

Svi ovi modeli imaju mehanizme kojima se omogućuje protok podataka te njihova obrada. Razdvojeni su na one koji upravljaju podatcima u podatkovnom sloju te one koji upravljaju podatcima u aplikacijskom sloju. Pri podatkovnom sloju cilj nije obrađivati podatke, već je cilj pružiti jedinstveno sučelje za izvršavanje operacija stvaranja, čitanja, ažuriranja i brisanja podataka, te vraćanje rezultata izvršavanja tih operacija. Rezultat čitanja bi bio tok (engl. *flow*) nekog tipa podatka, dok rezultati ostalih operacija mogu sadržavati povratni tip podatka zvan Rezultat (engl. *Result*) koji pruža mogućnost uvida u informaciju koja opisuje uspješnost ili neuspješnost provedbe operacije u bazi podataka u podatkovnom sloju. Uz samu informaciju o tome je li provedba operacije bila uspješna ili ne, moguće je i pružiti dodatne informacije poput nužnih informacija pri neuspjehu. Pri neuspjehu moguće je slati vlastite ili pak neke standardizirane tipove grešaka (engl. *error*) i time pružiti mogućnost da se unutar same greške pristupi njezinoj standardnoj poruci u slučaju kada bi se korisniku trebala prikazati ta poruka radi lakšeg navođenja korisnika u željenom smjeru ili radi lakšeg pronalaženja greške u sustavu radi njezinog uklanjanja.

Za Room bazu podataka i njezin model stavke u narudžbi koja nije potvrđena, pri podatkovnom sloju taj mehanizam upravljanja podatcima jest objekt za pristup podatcima (engl. *data access object*) koji pruža mogućnosti dohvaćanja svih stavki u nepotvrđenoj narudžbi, dodavanje i brisanje stavki, smanjivanje ili povećavanje količine stavki te brisanje stavke ukoliko pri smanjivanju stavke se broj stavki smanji na broj ispod brojke jedan. Osim samoga objekta za pristup podatcima, postoji i repozitorij (engl. *repository*) koji poziva taj objekt za pristup podatcima pružajući još jedan sloj apstrakcije.

Za Firestore bazu podataka ne postoji objekt za pristup podatcima, već repozitorij direktno obavlja poziv na Firestore instancu. Pri čitanju sa Firestore baze podataka vraća tok, uspostavlja komunikaciju u stvarnom vremenu putem slušatelja, te pretvara dobiveni dokument u model podatkovnog sloja. Pri ostalim operacijama vraća rezultat izvršenja operacije, time opisuje uspjeh ili neuspjeh radi daljnjeg rukovanja tim informacijama.

Repozitorij ugostiteljskih objekata pruža mogućnosti dobivanja podataka o svim ugostiteljskim objektima, dohvaćanja ugostiteljskog objekta po njegovoj identifikacijskoj oznaci te dodavanje ugostiteljskog objekta.

Repozitorij stavki na jelovniku pruža mogućnosti dobivanja podataka o svim stavkama jelovnika ugostiteljskog objekta, dodavanje stavke u jelovnik te brisanje stavke iz jelovnika.

Repozitorij narudžbi pruža mogućnosti dobivanja podataka o svim aktivnim te dovršenim narudžbama ugostiteljskog objekta i stavki povezanih s njima, dohvaćanje jedinstvene narudžbe po njezinoj identifikacijskoj oznaci, dodavanje narudžbi i stavki povezanih s narudžbom, ažuriranje podataka u narudžbi i stavki narudžbe te na posljetku brisanje narudžbi i stavki narudžbi.

Repozitorij osoblja pruža mogućnosti dobivanja podataka o svom osoblju, dohvaćanje podataka o jedinstvenoj instanci osoblja prema identifikacijskoj oznaci te korisničkom imenu i lozinki. Ujedno pruža mogućnost dohvaćanja podataka o svom osoblju određenog ugostiteljskog objekta, ali i odobreno i neodobreno osoblje zasebno. Moguće je dodati novu instancu osoblja, ažurirati podatke i obrisati instancu osoblja. Na posljetku postoji mogućnost dobivanja informacije o tome postoji li već određeno korisničko ime u bazi podataka.

U podatkovnom sloju lokalno na uređaju se čuvaju i podatci o trenutnom korisniku pomoću DataStore biblioteke. Pružaju se mogućnosti dodavanja podataka o trenutnom korisniku te brisanje podataka o trenutnom korisniku, odnosno stavljanje na zadanu vrijednost koja predstavlja nedostatak podataka o trenutnom korisniku.

Pri instanciranju (engl. *instantiation*) baze podataka, objekta za dohvaćanje podataka te repozitorija stvara se jednočlani objekt koji upotrebljavajući načelo jednočlanosti objektno orijentiranog programiranja daje mogućnost stvaranja jedinstvenog objekta radi učinkovite upotrebe raspoloživih sredstava sustava, poglavito memorijskih sredstava. Slanje te instance prema dijelovima sustava kojima je potrebna odvija se pomoću ubrizgavanja ovisnosti putem

Koin razvojnog okvira. Isto tako za vlastito instanciranje potrebne su im razne instance drugih dijelova sustava čije su im ovisnosti ubrizgane opet putem Koin razvojnog okvira.

4.2.2. Aplikacijski sloj

Aplikacijski sloj seže od podatkovnog sloja do prezentacijskog sloja aplikacije. Prvo gledajući vezu aplikacijskog sloja sa podatkovnim slojem nameće se struktura servisa (engl. *service*). Svaki pojedini repozitorij unutar podatkovnog sloja mora biti iskorišten unutar aplikacijskog sloja. U većini primjena, servisi služe samo kao dodatna apstrakcija podatkovnog sloja pružajući jednake mogućnosti kao i repozitoriji implementirajući metode repozitorija direktno bez dodatnog rukovanja podatcima, osim pretvaranja podataka iz modela baze na aplikacijski model [17].

Jedine značajne razlike se nalaze unutar servisa za narudžbe. Servis za narudžbe je odgovoran i za repozitorij narudžbi Firestore baze podataka i za repozitorij stavki narudžbi koje nisu potvrđene. Pri potvrđivanju narudžbe, servis stvara narudžbu, pretvara stavke nepotvrđene narudžbe iz Room baze podataka u stavke narudžbe u Firestore bazi podataka i povezuje stavke s narudžbom preko identifikacijske oznake narudžbe te na posljetku briše nepotvrđene stavke narudžbe jer su postale potvrđene i sada se nalaze na mrežnoj bazi podataka. Ovaj proces se smatra stvaranjem narudžbe i omogućuje ponovno unošenje nove narudžbe jer su svi lokalni podatci izbrisani. Uz to, pri dovršavanju narudžbe, poziva metodu repozitorija koja ažurira podatke narudžbe i dodaje informacije u polje koje sadrži identifikacijsku oznaku korisnika koji je dovršio narudžbu te označava narudžbu neaktivnom. Na posljetku, pri brisanju narudžbe mora se osigurati brisanje i same narudžbe i stavki narudžbe pojedinačno, tako da se te metode repozitorija pozivaju unutar jedne metode servisa.

Kao i u podatkovnom sloju, ubrizgavanje potrebnih ovisnosti unutar samih servisa se vrši putem Koin razvojnog okvira. Isto tako se stvaraju jednočlane instance svih servisa u pripremi za njihovo ubrizgavanje u ostale dijelove sustava gdje su potrebni.

Postoji popis (engl. *enum*) za uloge koje osoblje može imati i sadržava tri vrijednosti: administrator, konobar i ništa. Sve konstantne vrijednosti (engl. *constants*) koje se koriste unutar aplikacije na više mjesta i koje nisu usko vezane uz jedan koncept ili jednu datoteku, upisane su u datoteku naziva konstantne vrijednosti i od tamo se pozivaju i koriste gdje su potrebne.

Za svaki od modela u aplikacijskom sloju koji se mora prikazivati u prezentacijskom sloju potrebno je napraviti tzv. stanje prikaza (engl. *viewstate*), što je zapravo klasa podataka koja iz modela uzima samo one podatke koji su potrebni za prikaz. Za svako pretvaranje modela u stanje prikaza, potreban je maper (engl. *mapper*). Maper je klasa koji sadrži logiku izvlačenja potrebnih podataka za prikaz iz modela i sprema ih u stanje prikaza osiguravajući učinkovitu upotrebu raspoloživih sredstava sustava. Time se omogućava brži i manje memorijski zahtjevan proces ažuriranja stanja na ekranu.

4.2.3. Navigacijski sloj

U navigacijskom sloju se unutar zatvorene klase (engl. *sealed class*) koja sadrži podatkovne objekte (engl. *data object*) stvaraju putanje i oznake za svako odredište, odnosno svaki ekran za koji nisu potrebni parametri. Navigacija se odvija između sljedećih odredišta:

- Odabir stavki za narudžbu
- Potvrđivanje narudžbe
- Pregled aktivnih narudžbi
- Pregled dovršenih narudžbi (povijest narudžbi)
- Prijava
- Registracija
- Odobreno osoblje
- Neodobreno osoblje
- Stvaranje stavke u jelovniku
- Brisanje stavke u jelovniku.

Postoje tri odredišta za čije stvaranje je potreban parametar identifikacijska oznaka:

- Pojedinačno osoblje
- Pojedinačna aktivna narudžba
- Pojedinačna dovršena narudžba.

Za svako od ovih odredišta potrebna je identifikacijska oznaka kako bi se pristupilo podatcima za pojedinačni model u svrhu prikazivanja sadržaja i upravljanja tim modelom.

4.2.4. Prezentacijski sloj

Radi dobre uspostave temelja prezentacijskog sloja, potrebno je definirati teme, oblike, tipografiju, boje te tekst kako bi svaki element prezentacijskog sloja izgledao ujednačeno s ostatkom aplikacije. Ujedno se tako i postavlja standard projekta. Osim toga, u projektima koji to zahtijevaju, pri promjeni teme vrlo lako je moguće dodati nove stilove na postojeće i promijeniti ih jednim dodirom ekrana. Glavna pozadinska boja jest siva, dok je glavna boja teksta zelena.

Nakon dodavanja snažnih temelja koji su otporni na utjecaje dijelova aplikacije koji nisu direktno vezani uz same teme, nalaze se ekrani i njihove komponente. Ekrani koji su potrebni za projekt su jednaki odredištima iz navigacijskih odredišta. Gotovo svi ekrani imaju svoje stanje prikaza i model prikaza. Za svaku klasu modela prikaza postoji modul Koin razvojnog okvira koji stvara njezinu instancu putem ubrizgavanja ovisnosti te radi rukovanja ubrizgavanjem ovisnostima same instance unutar drugih instanci klasa u daljnjim koracima rada programa.

5. IZRADA APLIKACIJE

U ovom poglavlju objašnjena je implementacija dizajna sustava definiranog u prethodnom poglavlju uz prikaz i opis konačnog dizajna korisničkog sučelja i korisničkog iskustva pri korištenju aplikacije.

5.1. Izrada podatkovnog sloja

Linija ***Kod***

```
1:      private const val APP_DATABASE_NAME = "app_database.db"
2:      val databaseModule = module {
3:          single {
4:              Room.databaseBuilder(
5:                  androidApplication(),
6:                  OrderManagerDatabase::class.java,
7:                  APP_DATABASE_NAME,
8:              ).build()
9:          }
10:     }
```

Programski kod 5.1. Prikaz Koin modula za Room bazu podataka

Baze podataka se stvaraju preko modula Koin razvojnog okvira.

Linija ***Kod***

```
1:      val firebaseModule = module {
2:          single {
3:              FirebaseFirestore.getInstance()
4:          }
5:      }
```

Programski kod 5.2. Prikaz Firebase modula za Firestore bazu podataka

Kod Room baze podataka stvara se jednočlana instanca preko graditelja.

Firebase Firestore baza podataka stvara se putem načina definiranog unutar Firebase biblioteke.

Modeli su implementirani putem podatkovne klase (engl. *data class*) u Kotlin programskom jeziku.

Linija Kod

```
1:      data class Order(
2:          val id: String = "placeholder",
3:          val establishmentId: String,
4:          val tableNumber: String,
5:          val createOrderStaffId: String,
6:          val completeOrderStaffId: String,
7:          val active: Boolean,
8:          val createdAt: Timestamp = Timestamp.now(),
9:      ) {
10:         fun toDbOrder() =
11:             DbOrder(
12:                 id = id,
13:                 establishmentId = establishmentId,
14:                 tableNumber = tableNumber,
15:                 createOrderStaffId = createOrderStaffId,
16:                 completeOrderStaffId = completeOrderStaffId,
17:                 active = active,
18:                 createdAt = createdAt,
19:             )
20:     }
```

Programski kod 5.3. Prikaz aplikacijskog modela za narudžbu

Podatkovna klasa modela daje predložak za kreiranje modela. Glavne funkcionalnosti aplikacijskih modela i svih modela za Firestore bazu podataka su određivanje podataka koji se spremaju u instancu podatkovne klase i njihovih tipova te metoda koja omogućuje jednostavno pretvaranje modela iz aplikacijskog modela u model baze podataka i obrnuto. Dok su podatkovne klase koje opisuju modele aplikacije i modele Firestore baze podataka gotovo jednake, razlika se pojavljuje kod podatkovne klase koja opisuje model Room baze podataka.

Linija Kod

```
1:      data class NotConfirmedOrderItem(
2:          val id: Int = 0,
3:          val name: String,
4:          val amount: Int,
5:      ) {
6:          fun toDbNotConfirmedOrderItem() =
7:              DbNotConfirmedOrderItem(
8:                  id = id.toLong(),
9:                  name = name,
10:                 amount = amount,
11:             )
12:     }
```

Programski kod 5.4. Prikaz modela Room baze podataka za nepotvrđenu narudžbu

Model Room baze podataka uz sve što sadrže aplikacijski model i model Firestore baze podataka sadrži i ime tablice unutar anotacije (engl. *annotation*) koja označava entitet (engl. *entity*), a i definiran je i primarni ključ (engl. *primary key*) koji jednoznačno označava instancu te se automatski povećava svakom novom instancom podatkovne klase.

Objekt za pristup podatcima koji je potreban za rad s Room bazom podataka implementiran je putem sučelja s Dao (engl. *data access object*) anotacijom.

Linija Kod

```
1:        @Dao
2:        interface NotConfirmedOrderItemDao {
3:            @Query("SELECT * FROM orderItems")
4:            fun getOrderItems(): Flow<List<DbNotConfirmedOrderItem>>
5:
6:            @Insert(onConflict = OnConflictStrategy.REPLACE)
7:            fun insertOrderItem(orderItem: DbNotConfirmedOrderItem)
8:
9:            @Query("DELETE FROM orderItems")
10:           suspend fun deleteAllOrderItems(): Int
11:
12:           @Query("UPDATE orderItems SET amount=amount+1 WHERE
          name=:name")
13:           fun incrementOrderItemAmount(name: String)
14:
15:           @Query("UPDATE orderItems SET amount=amount-1 WHERE id=:id")
16:           fun subtractOrderItemAmount(id: Int)
17:
18:           @Query("DELETE FROM orderItems WHERE id=:id AND amount=0")
19:           fun deleteOrderItemIfNecessary(id: Int)
20:       }
```

Programski kod 5.5. Objekt za pristup podatcima

Postoje dvije vrste anotacija na metodama sučelja. Jedna anotacija omogućuje bilo kakve SQL upite, a druga omogućuje pisanje u bazu podataka s funkcionalnošću zamjene redaka pri sukobu postojećeg retka i retka koji se dodaje. Metode primaju ili vraćaju podatke. Pri dohvaćanju zbirke podataka, zbirka je umotana u tok što omogućava reagiranje sustava na promjene u zbirci podataka u stvarnom vremenu, odnosno rad s najnovijim dostupnim podatcima. Jedina druga metoda koja ima povratni tip podatka jest metoda koja briše sve retke te vraća broj izbrisanih redaka. Druge metode primaju ili identifikacijsku oznaku ili naziv nedovršene stavke kako bi pronašli redak i izvršile neki posao nad tim retkom ili pak primaju instancu modela kako bi

stvorile novi redak prema podatcima iz instance. SQL upiti koje se koriste su upiti za odabiranje, brisanje ili ažuriranje redaka u tablici sa dodatnim uvjetima tamo gdje su potrebni.

Sljedeći stupanj apstrakcije, odnosno implementacije koja će biti skrivena od drugih slojeva su repozitoriji.

Metode repozitorija za Room bazu podataka imaju već skrivenu implementaciju za direktnu komunikaciju s bazom podataka te pružaju sloj koji u svojoj implementaciji treba samo pozvati metode objekta za pristupanje podatcima i predati mu podatke ili primiti podatke od njega.

Linija Kod

```
1:      override fun orderedItems(): Flow<List<DbNotConfirmedOrderItem>> =
2:          orderedItemDao.getOrderItems().map {
3:              it.map { dbOrderedItem ->
4:                  DbNotConfirmedOrderItem(
5:                      id = dbOrderedItem.id,
6:                      name = dbOrderedItem.name,
7:                      amount = dbOrderedItem.amount,
8:                  )
9:              }
10:         }.shareIn(
11:             scope = CoroutineScope(bgDispatcher),
12:             started = SharingStarted.WhileSubscribed(1000L),
13:             replay = 1,
14:         )
15:
16:     override suspend fun addOrderedItem(orderedItem:
        DbNotConfirmedOrderItem) {
17:         withContext(bgDispatcher) {
18:             orderedItemDao.insertOrderItem(
19:                 orderedItem
20:             )
21:         }
22:     }
```

Programski kod 5.6. Repozitorij Room baze podataka

Na programskom kodu 5.6. prikazane su dvije metode od kojih jedna poziva metodu objekta za pristupanje podatcima koja dohvaća zbirku svih stavki koje nisu potvrđene te razotkriva podatke za daljnju upotrebu. Druga metoda dodaje novu stavku koja nije potvrđena tako što poziva metodu objekta za pristupanje podatcima i predaje model koji je njoj samoj predao neki drugi sloj. Pošto je skrivanje implementacije već odrađeno, posao repozitorija Room baze podataka jest pozvati već implementirane metode, što znači da taj posao nije kompliciran i vrlo je izravan.

Kod Firebase Firestore baze podataka, repozitorij je prva točka gdje se implementiraju funkcionalnosti komuniciranja s bazom podataka, stoga je proces mnogo kompliciraniji od samog pozivanja metode.

Linija Kod

```
1:      override fun getAllEstablishments(): Flow<List<DbEstablishment>> {
2:          return callbackFlow {
3:              val listenerRegistration =
        fireStore.collection(FIRESTORE_COLLECTION_ESTABLISHMENTS)
4:                  .addSnapshotListener { snapshot, error ->
5:                      try {
6:                          if (error != null) {
7:                              close(error)
8:                              return@addSnapshotListener
9:                          }
10:                         if (snapshot != null) {
11:                             val establishmentCollection =
12:
        snapshot.documents.map(::mapEstablishmentDocumentToDbEstablishment)
13:                             try {
14:                                 trySend(establishmentCollection)
15:                             } catch (sendException: Exception) {
16:                                 Log.e("getAllEstablishments", "Failed to
        send data", sendException)
17:                             }
18:                         }
19:                     } catch (snapshotException: Exception) {
20:                         Log.e("getAllEstablishments", "SnapshotListener
        error", snapshotException)
21:                     }
22:                 }
23:             awaitClose { listenerRegistration.remove() }
24:         }
25:     }
```

Programski kod 5.7. Repozitorij Firestore baze podataka, tablice ugostiteljski objekti, dohvaćanje ugostiteljskih objekata

Pri dohvaćanju redaka iz tablice, ručno se stvara tok, poziva se zbirka te se na nju stavlja slušatelj. Slušatelj omogućava dobivanje najnovijeg stanja u tablici na način da se pri izmjeni

podataka unutar tablice obavijeste svi slušatelji da se promjena dogodila, a onda slušatelji reagiraju na promjene ažurirajući stanja unutar aplikacije. Dodani su dodatni rukovatelji iznimkama putem bloka koji pokuša odraditi zadatak i hvata iznimku koja se može dogoditi pri pokušaju odrađivanja tog zadatka.

Linija Kod

```
1:      private fun mapEstablishmentDocumentToDbEstablishment(establishment:
        DocumentSnapshot): DbEstablishment {
2:          return DbEstablishment(
3:              id = establishment.id,
4:              name = establishment.getString("name") ?: ""
5:          )
6:      }
```

Programski kod 5.8. Metoda mapiranja

Pošto je tip podatka koji se vraća pri dohvaćanju zbirke podataka snimak datoteke (engl. *document snapshot*), a taj tip nije pogodan za rukovanje unutar aplikacije, potrebno ga je pretvoriti u model baze podataka putem privatne metode. Metoda je privatna jer je potrebna samo unutar klase u kojoj se koristi, stoga se prati načelo učahurivanja (engl. *encapsulation*). Kada slušatelj više nije potreban, on se uklanja kako bi se smanjila opterećenja na sredstvima sustava.

Linija Kod

```
1:      override suspend fun getEstablishmentById(establishmentId: String):
        DbEstablishment? {
2:          val establishmentDocument =
        fireStore.collection(FIRESTORE_COLLECTION_STAFF)
3:              .document(establishmentId)
4:              .get()
5:              .await()
6:
7:          if (!establishmentDocument.exists()) {
8:              return
        mapEstablishmentDocumentToDbEstablishment(establishmentDocument)
9:          }
10:         return null
11:     }
12:
13:     override suspend fun addEstablishment(establishment:
        DbEstablishment) {
14:         fireStore.collection(FIRESTORE_COLLECTION_STAFF)
15:             .add(establishment)
16:             .addOnFailureListener {
17:                 Log.d("Something went wrong: ", it.message.toString())
18:             }
19:     }
```

Programski kod 5.9. Repozitorij Firestore baze podataka, tablice ugostiteljski objekti, dohvaćanje po identifikacijskoj oznakci i dodavanje ugostiteljskih objekata

Pri provedbi drugih zadataka koji ne podrazumijevaju rukovanje podatcima u stvarnom vremenu slušatelji koji su aktivni na duže vrijeme nisu potrebni. Postoje mnogi načini provjere dostupnosti podataka i uspješnosti izvršavanja zadatka. Na programskom kodu 5.9. moguće je vidjeti uporabu funkcije koja provjerava postoji li zatraženi objekt u bazi podataka ili pak stvaranje jednokratnog slušatelja koji će javiti je li se zadatak uspio izvršiti ili ne, uz to omogućavajući izvršavanje određenih zadataka u oba slučaja.

Linija Kod

```
1:      override suspend fun getNotApprovedStaff(establishmentId: String):
        Flow<List<DbStaff>> {
2:          return callbackFlow {
3:              val listenerRegistration =
4:                  fireStore
5:                      .collection(FIRESTORE_COLLECTION_STAFF)
6:                      .whereEqualTo("establishmentId", establishmentId)
7:                      .whereEqualTo("approved", false)
8:                      .orderBy("role", Query.Direction.ASCENDING)
9:                      .addSnapshotListener { snapshot, error ->
10:                         try {
11:                             if (error != null) {
12:                                 close(error)
13:                                 return@addSnapshotListener
14:                             }
15:                             if (snapshot != null) {
16:                                 val staffCollection =
17:
        snapshot.documents.map(::mapStaffDocumentToDbStaff)
18:                                 try {
19:                                     trySend(staffCollection)
20:                                 } catch (sendException: Exception) {
21:                                     Log.e("getAllStaff", "Failed to send
        data", sendException)
22:                                 }
23:                             }
24:                         } catch (snapshotException: Exception) {
25:                             Log.e("getAllStaff", "SnapshotListener
        error", snapshotException)
26:                         }
27:                     }
28:             awaitClose { listenerRegistration.remove() }
29:         }
30:     }
```

Programski kod 5.10. Repozitorij Firestore baze podataka, tablice osoblje, dohvaćanje neodobrenog osoblja ugostiteljskog objekta

Na programskom kodu 5.10. moguće je vidjeti i na koji način se dodaju upiti za bazu podataka kada je u slučaju Firebase Firestore. Za gdje (engl. *where*) uvjet koristi se whereEqualTo metoda, za poredaj po (engl. *order by*) uvjet koristi se orderBy metoda i postoji još mnogo metoda koje implementiraju funkcionalnosti SQL upita radi standardiziranja tih upita u Kotlin programskom jeziku pružajući jedinstveni način komunikacije s bazom podataka.

Linija Kod

```
1:      override suspend fun getStaffById(id: String): Result<DbStaff> {
2:          return try {
3:              val staffDocument =
        fireStore.collection(FIRESTORE_COLLECTION_STAFF)
4:                  .document(id)
5:                  .get()
6:                  .await()
7:              if (staffDocument.exists()) {
8:                  Result.success(mapStaffDocumentToDbStaff(staffDocument))
9:              } else {
10:                 Result.failure(StaffNotFoundException())
11:             }
12:         } catch (exception: Exception) {
13:             Result.failure(FirestoreException(exception))
14:         }
15:     }
```

Programski kod 5.11. Repozitorij Firestore baze podataka, tablice osoblje, dohvaćanje instance osoblja po identifikacijskoj oznaci

Na programskom kodu 5.11. prikazano je stvaranje Rezultat klase koja daje mogućnost omotavanja nekog tipa podatka i davanja informacije o tome je li zadatak uspio ili ne. Pri uspjehu na Rezultat klasi koristi se metoda success i njoj se preda podatak koji Rezultat klasa omotava. Pri neuspjehu koristi se metoda failure i njoj se preda iznimka kojoj se kasnije može pristupati kako bi se dobila njezina poruka ili iskoristila bilo kakva druga funkcionalnost iznimke.

Moguće je stvarati vlastite iznimke, tako su u ovom projektu stvorene iznimke za sljedeće situacije:

- Firestore iznimka
- Prazno polje pri upisivanju podataka
- Narudžba nije pronađena
- Instanca osoblja nije pronađena
- Korisničko ime već postoji
- Prekratka duljina lozinke
- Preslaba lozinka.

Linija Kod

```
1:     class WeakPasswordException : Exception("Password too weak.")
```

Programski kod 5.12. Iznimka za preslabu lozinku

Iznimke koje su stvorene s ciljem da sadrže samo poruku kojoj bi se pristupalo, napravljene su na način da implementiraju Exception klasu Kotlin jezika koja prima poruku iznimke. Firestore iznimka je jedina iznimka koja prima instancu druge iznimke jer Firestore stvara vlastitu iznimku, stoga ju je potrebno samo proslijediti dalje kako bi sve funkcionalnosti bile dostupne.

Pri prijavi korisnika podatci o trenutnom korisniku lokalno se pohranjuju u DataStore. Ti lokalno pohranjeni podatci su dostupni u kontekstu aplikacije.

Linija Kod

```
1:      object UserDataSerializer : Serializer<UserData> {
2:          override val defaultValue: UserData
3:              get() = UserData()
4:
5:          override suspend fun readFrom(input: InputStream): UserData {
6:              return try {
7:                  Json.decodeFromString(
8:                      deserializer = UserData.serializer(),
9:                      string = input.readBytes().decodeToString(),
10:                 )
11:             } catch (exception: SerializationException) {
12:                 exception.printStackTrace()
13:                 defaultValue
14:             }
15:         }
16:
17:         override suspend fun writeTo(t: UserData, output: OutputStream)
        {
18:             output.write(
19:                 Json.encodeToString(
20:                     serializer = UserData.serializer(),
21:                     value = t,
22:                 ).encodeToByteArray()
23:             )
24:         }
25:     }
```

Programski kod 5.13. Serijalizator za korisničke podatke trenutnog korisnika

Programski kod 5.13. prikazuje serijalizaciju (engl. *serialization*) i deserijalizaciju (engl. *deserialization*) korisničkih podataka trenutnog korisnika omogućavajući čitanje iz izlaznog toka (engl. *output stream*) i upisivanje u ulazni tok (engl. *input stream*) podataka. Serijalizator (engl. *serializer*) je u ovom slučaju klasa koja omogućava pretvaranje objekta koji sadrži korisničke

podatke u JSON format i obrnuto radi omogućavanja komunikacije spremišta podataka i aplikacije te međusobnog prijenosa podatka [18].

5.2. Izrada aplikacijskog sloja

Nakon opisa sustava direktne komunikacije sa bazama podataka i spremištima podataka, slijedi opis komunikacije same aplikacije s tim sustavom. Aplikacijski dio rukovanja podatcima je implementiran putem servisa. Metode servisa većinom samo pozivaju metode repozitorija i pretvaraju modele baze podataka u aplikacijske modele i obrnuto te nemaju dodatne funkcionalnosti osim tih, već služe za odvajanje aplikacijskog i podatkovnog sloja. Ali u nekim slučajevima, tamo gdje je to potrebno, pošto repozitoriji imaju jednostavne i gotovo atomske operacije (engl. *atomic operations*) nad bazom podataka, kompleksni procesi se ostavljaju servisima. Servisi koriste nekoliko metoda iz repozitorija kako bi obavili zadatke koji nisu najjednostavnije vrste.

Linija Kod

```
override suspend fun confirmOrder(order: Order): Result<Unit> {
    try {
        println("Starting confirmOrder function")

        val orderId = orderRepository.addOrder(order.toDbOrder())
        println("Order added successfully with orderId: $orderId")

        val orderItems = notConfirmedOrderRepository.orderedItems()
        processOrderItems(orderId.getOrNull()!!,
orderItems.firstOrNull()!!)

        println("All order items added successfully")
        notConfirmedOrderRepository.deleteAllOrderItems()

        println("Finished confirmOrder function")
        return Result.success(Unit)
    } catch (exception: Exception) {
        println("Caught exception: ${exception.message}")
        return Result.failure(exception)
    } finally {
        scope.cancel() // Cancel the CoroutineScope when done
    }
}
```

Programski kod 5.14. Metoda servisa narudžbi koja implementira potvrđivanje narudžbe

Programski kod 5.14. jedan je od primjera koji objašnjava razlog zašto arhitektura servisa postoji i zašto nije dovoljno imati samo repozitorije. Proces potvrđivanja narudžbe se odvija na način da se prvo stvori narudžba i usput se dohvati identifikacijska oznaka narudžbe, zatim se pojedine stavke u narudžbi procesuiraju (engl. *processing*) i izbrišu se sve lokalno spremljene narudžbe. Cijeli proces popraćen je rukovanjem iznimkama i njihovim izlaganjem sloju koji poziva metodu servisa.

Linija Kod

```
1:      private suspend fun processOrderItems(
2:          orderId: String,
3:          orderItems: List<DbNotConfirmedOrderItem>
4:      ) {
5:          val deferredResults = orderItems.map { orderItem ->
6:              val dbOrderItem = orderItem.toOrderItem(orderId =
        orderId).toDbOrderItem()
7:              scope.async {
8:                  orderRepository.addOrderItem(dbOrderItem)
9:              }
10:         }
11:
12:         // Await each individual deferred result
13:         deferredResults.forEach { deferred ->
14:             val result = deferred.await()
15:             result.fold(
16:                 onSuccess = {
17:                     println("Order item added successfully")
18:                 },
19:                 onFailure = { exception ->
20:                     println("Failed to add order item:
        ${exception.message}")
21:                     throw exception
22:                 }
23:             )
24:         }
25:     }
```

Programski kod 5.15. Metoda servisa narudžbi procesuira stavke narudžbe

Metoda prikazana programskim kodom 5.15. pobliže pokazuje što se događa sa procesuiranjem zbirke stavki narudžbe koja nije potvrđena. Od svake stavke narudžbe koja nije potvrđena stvori se nova stavka narudžbe te se za tu stavku narudžbe preda i identifikacijska oznaka narudžbe

kojoj ta stavka pripada, a ostali podatci se dobiju iz nepotvrđene stavke. Proces je opet popraćen rukovanjem iznimkama.

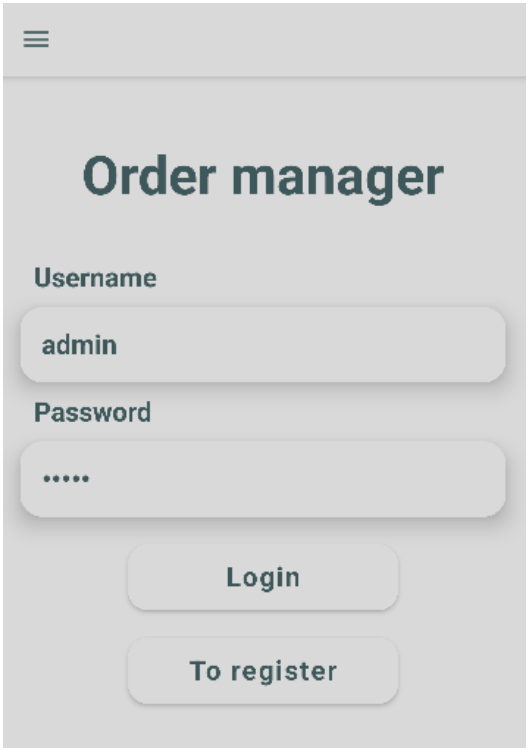
Kao i potvrđivanje narudžbe, brisanje narudžbe isto sadrži mnoštvo malih zadataka koji implementiraju funkcionalnost brisanja same narudžbe i brisanje pojedinačnih stavki unutar te iste narudžbe što je proces kompliciraniji od jednostavnog pristupanja bazi s jednom operacijom.

5.3. Izrada prezentacijskog sloja

Prezentacijski sloj većinski se sastoji od ekrana, komponenti te stanja prikaza i klasa modela prikaza. Ovo poglavlje raščlanjeno je po konceptima od kojih neki može sadržavati više ekrana, komponenti, objekata stanja prikaza i klasa modela prikaza.

5.3.1. Prijava korisnika

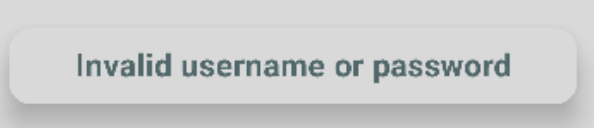
Prijava korisnika dostupna je svakom korisniku koji još nije prijavljen.



Slika 5.1. Ekran prijave korisnika

Na ovom ekranu, korisnik koji već ima svoj korisnički račun može se prijaviti u sustav. Proces prijave odvija se na način da korisnik upiše svoje korisničko ime u prvo polje za unos, a lozinku u drugo polje za unos. Klikom gumba Login korisnik šalje svoje podatke na autentifikaciju.

Proces autentifikacije uključuje provjeru postoji li korisnik s tim korisničkim imenom i lozinkom u sustavu, ako postoji, onda se podatci tog korisnika spreme u podatke trenutnog korisnika aplikacije te aplikacija odradi navigaciju na različite ekrane po ulogama. Korisnik s ulogom administrator navigira se na ekran odobrenog osoblja, korisnik sa ulogom konobar navigira se na ekran odabira stavki za narudžbu, a korisnik koji nije uspješno autentificiran (engl. *authenticated*) dobiva obavijest.



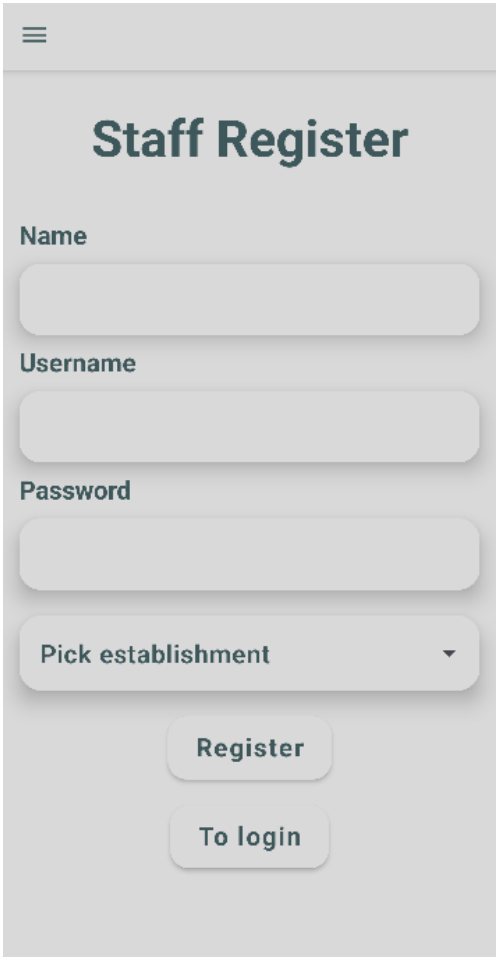
Slika 5.2. Poruka pri netočno unesenim podatcima

Slika 5.2. prikazuje obavijest koju dobiva korisnik koji nije uspješno autentificiran. Poruka je implementirana preko Snackbar elementa. Svakom novom neuspješnom prijavom dok je Snackbar aktivan, on se deaktivira te se na njegovo mjesto stavlja drugi, kako bi naznačio reagiranje sustava na korisnikov unos. Dužina trajanja Snackbar elementa stavljena je na dugačko.

Klikom na gumb To register aplikacija navigira korisnika na ekran za registraciju.

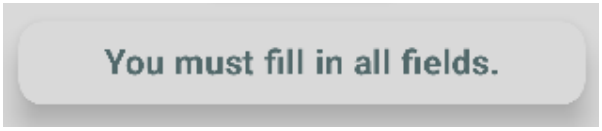
5.3.2. Registracija korisnika

Registracija korisnika dostupna je svakom korisniku koji još nije prijavljen.



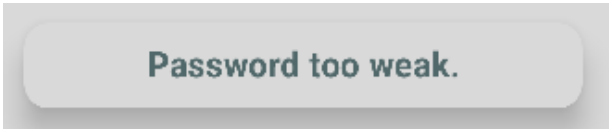
Slika 5.3. Ekran registracije korisnika

Ovaj ekran namijenjen je korisnicima koji nemaju korisnički račun pri ustanovi, odnosno novim zaposlenicima. Novi korisnici moraju napisati svoje ime, korisničko ime i lozinku. Uz to moraju odabrati ugostiteljski objekt iz padajućeg izbornika ugostiteljskih objekata. Pri registraciji, podatci prolaze kroz proces koji određuje njihov integritet. Postoji nekoliko mogućih poruka koje korisnik može dobiti pri kliku gumba Register koji okida proces registracije.

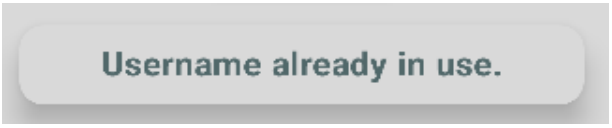


Slika 5.4. Poruka pri ne popunjenim poljima

Slika 5.5. Poruka pri prekratkoj lozinci



Slika 5.6. Poruka pri slaboj lozinci

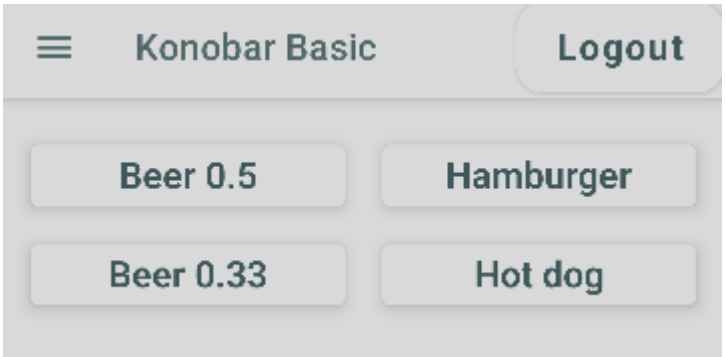


Slika 5.7. Poruka pri ne jedinstvenom korisničkom imenu

Poruka sa slike 5.4. pojavljuje se ukoliko korisnik nije popunio jedno ili više polja, ili nije odabrao ugostiteljski objekt. Slika 5.5. naznačuje kako je minimalna duljina lozinke 8 znakova. Slika 5.6. ističe poruku pri lozinki koja je slaba, odnosno ne sadrži barem jedan element od svakog iz grupe: mala slova, velika slova, brojka i simbol. Na posljetku slika 5.7. daje korisniku do znanja da je korisničko ime koje je odabrao već u upotrebi.

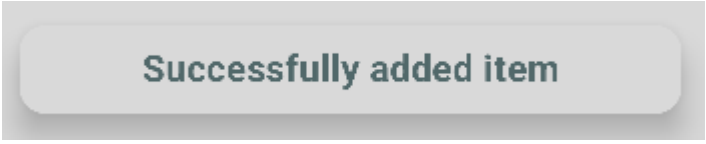
5.3.3. Stvaranje narudžbe

Stvaranje narudžbe dostupno je ulozi konobar.



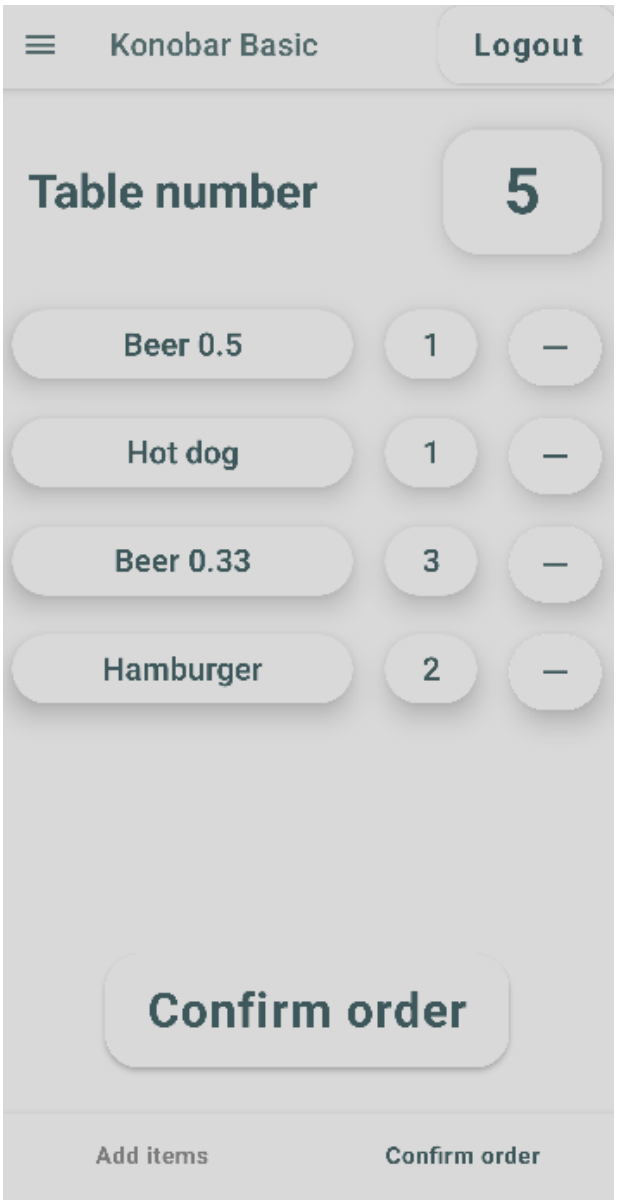
Slika 5.8. Ekran odabira stavki za narudžbu

Na ekranu odabira stavki za narudžbu moguće je kliknuti na stavku kako bi se dodala u narudžbu koja se trenutno stvara, odnosno narudžbu koja nije potvrđena.



Slika 5.9. Obavijest pri dodavanju stavke

Klikom na bilo koju stavku pojavljuje se obavijest kako je stavka dodana u narudžbu radi obavještavanja korisnika o uspješnom dodavanju. Uz obavijest obavlja se proces dodavanja stavke ukoliko ta stavka već nije dodana u narudžbu, a ukoliko jest, poveća se količina te stavke za jedan.

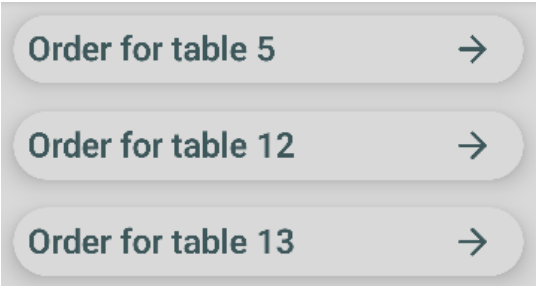


Slika 5.10. Ekran za potvrđivanje narudžbe

Na ekranu za potvrđivanje narudžbe, do kojeg se navigira klikom na donju traku, upisuje se broj stola kojemu je narudžba namijenjena te se može smanjivati količina stavki u narudžbi. Ukoliko se količina smanji na broj ispod brojke jedan, stavka se kompletno uklanja iz narudžbe. Klikom na gumb Confirm order narudžba se potvrđuje, upisuje u mrežnu bazu podataka te se sve stavke te narudžbe uklone iz lokalne baze podataka kako bi se mogle unositi nove stavke. Pri tome, korisnika se navigira na ekran odabira stavki u narudžbi.

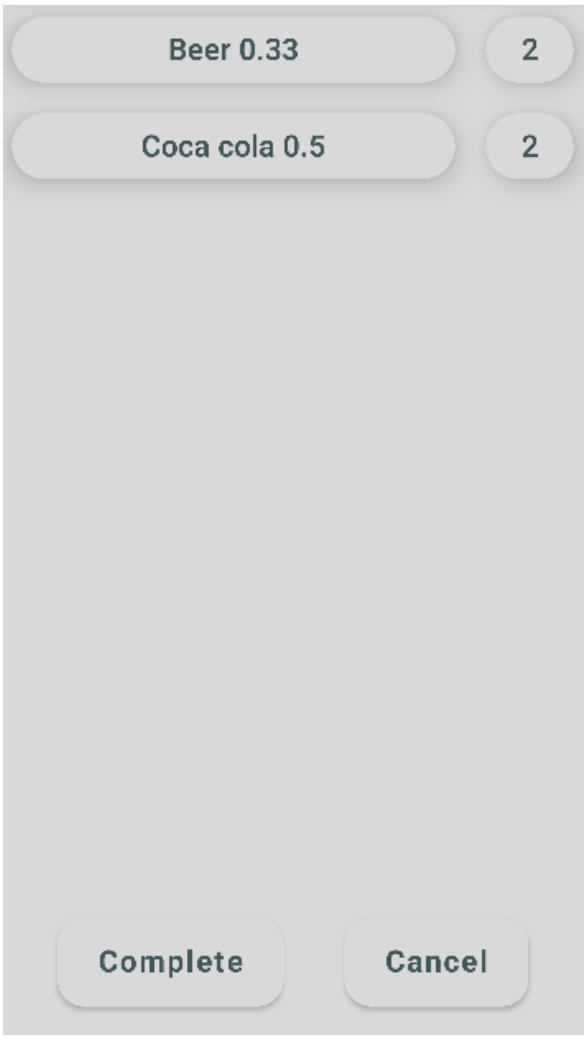
5.3.4. Pregled narudžbi

Pregled narudžbi dostupan je ulozi administrator i ulozi konobar.



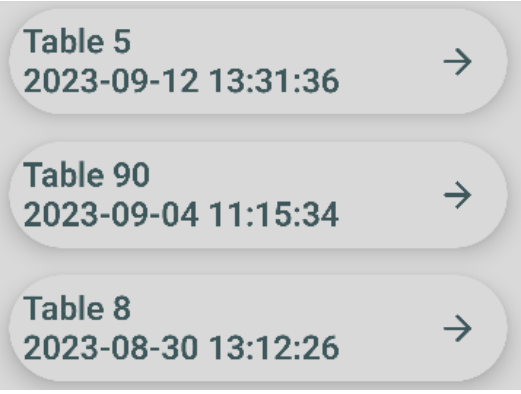
Slika 5.11. Ekran za pregled aktivnih narudžbi

Na ekranu za pregled aktivnih narudžbi nalaze se elementi koji predstavljaju narudžbe. Elementi se razlikuju po broju stola. Klikom na elemente otvara se ekran s detaljima o narudžbi.



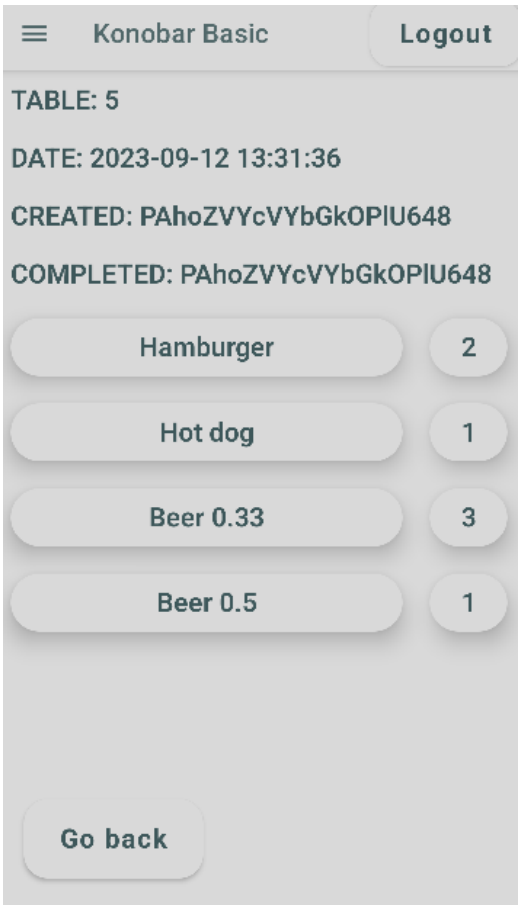
Slika 5.12. Ekran s detaljima o aktivnoj narudžbi

Ekran sa slike 5.12. prikazuje stavke u narudžbi te nudi dvije mogućnosti: dovrši i otkaži narudžbu. Dovršetkom narudžbe, narudžba se dodaje u arhivu dovršenih narudžbi, a otkazivanjem narudžbe, narudžba se briše iz sustava. Klikom bilo kojeg od gumbova korisnik se vraća na ekran s aktivnim narudžbama.



Slika 5.13. Ekran za pregled dovršenih narudžbi

Na ekranu za pregled dovršenih narudžbi jedina razlika od ekrana za pregled aktivnih narudžbi su datum i vrijeme kreiranja narudžbe koji su vidljivi uz broj stola. Datum i vrijeme su važni kako bi se narudžba mogla lakše kategorizirati i locirati. Klikom na bilo koji od elemenata otvara se ekran s detaljima o pojedinoj dovršenoj narudžbi.

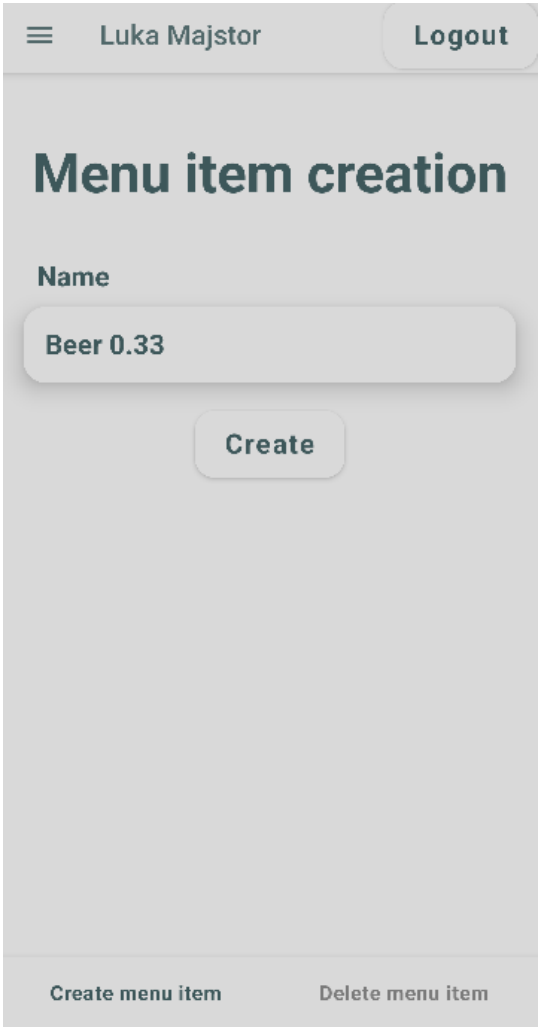


Slika 5.14. Ekran s detaljima o dovršenoj narudžbi

Na slici 5.14. prikazuje se ekran na kojemu se vidi broj stola, datum i vrijeme stvaranja narudžbe, identifikacijska oznaka korisnika koji je izdao i korisnika koji je dovršio narudžbu kao i same stavke narudžbe. U slučaju prigovora gostiju, narudžba se može lako identificirati. Pritiskom na gumb Go back, korisnik se vraća na ekran za pregled dovršenih narudžbi.

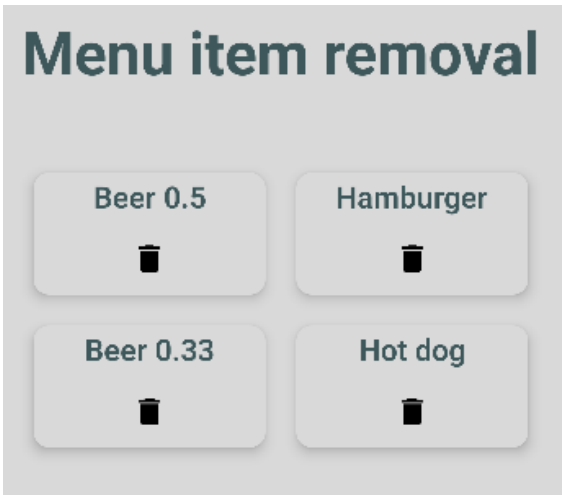
5.3.5. Upravljanje jelovnikom

Upravljanje jelovnikom dostupno je ulozi administrator.



Slika 5.15. Ekran za stvaranje stavki u jelovniku

Na ekranu sa slike 5.15. moguće je stvoriti novu stavku u jelovniku upisujući ime stavke te klikom na gumb Create.

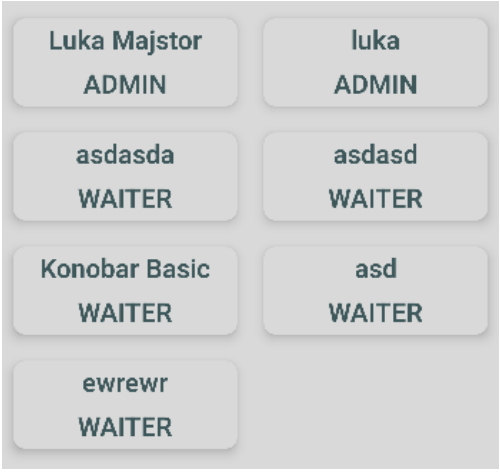


Slika 5.16. Ekran za brisanje stavki u jelovniku

Na ekranu za brisanje stavki u jelovniku moguće je vidjeti sve stavke u jelovniku koje su dostupne konobarima za dodavanje u narudžbe. Klikom na ikonu koja simbolizira kantu za smeće moguće je izbrisati stavku iz jelovnika pri čemu se dobije obavijest putem Snackbar elementa kako je stavka uspješno uklonjena iz jelovnika.

5.3.6. Upravljanje osobljem

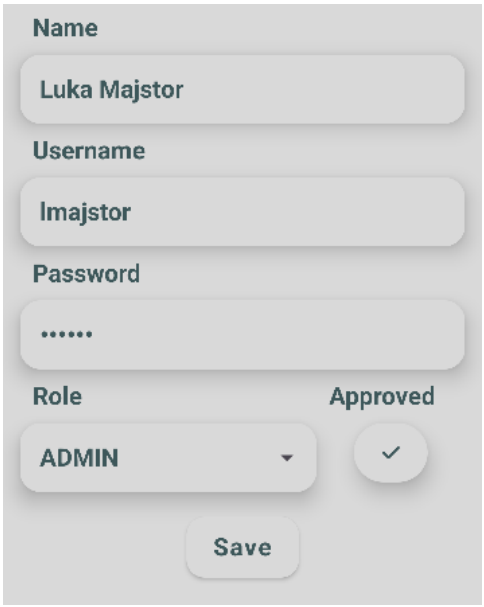
Upravljanje osobljem je dostupno ulozi administrator.



Slika 5.17. Ekran za pregled odobrenog osoblja

Ekrani za pregled odobrenog i neodobrenog osoblja su jednaki s razlikom da su drugi korisnici na svakom od ekrana, na jednom se nalaze oni odobreni, dok se na drugom nalaze oni koji nisu

odobreni. Kartica jednog korisnika prikazuje korisnikovo ime te njegovu ulogu. Klikom na karticu otvara se ekran za upravljanje korisničkim podatcima pojedinog korisnika.



Slika 5.18. Ekran za pregled i upravljanje podatcima korisnika

Na slici 5.18. prikazan je ekran koji omogućava mijenjanje svih podataka korisnika osim datuma stvaranja računa. Pošto svaki korisnik pri registraciji ima ulogu konobar, prilično je korisno administratoru imati ekran na kojemu može postaviti drugog korisnika administratorom. Isto tako na ovom ekranu se korisnički računi i odobravaju ili se odobrenje uklanja klikom na gumb ispod riječi Approved. Gumb prikazuje kvačicu za odobrene korisnike, a simbol x za korisnike koji nisu odobreni. Klikom na gumb Save podatci se spremaju. Ako se ne stisne gumb Save, podatci se ne mijenjaju i ponovnim ulaskom na ekran vraćaju se u prvotno stanje. Pri kliku gumba Save provodi se proces koji osigurava da su svi podatci prisutni, odnosno sva polja popunjena, korisničko ime jedinstveno te lozinka zadovoljava sve uvjete. Nakon procesa, korisnika se obavještava o uspješnosti odnosno neuspješnosti postupka i razlogom putem Snackbar elementa.

6. ZAKLJUČAK

Cilj ovog završnog rada jest stvoriti Android aplikaciju koja će omogućiti upravljanje narudžbama u ugostiteljskom objektu. Aplikacija koristi raznoliku lepezu tehnologija pomoću kojih implementira ideje koje se u stvarnom svijetu koriste pri razvoju aplikacija. Primjerice, ako ne postoji dobro razrađeni dizajn, kako i arhitekturni, tako i vizualni, proces stvaranja aplikacije je za veću mjeru teži te dulje traje. U stvaranju, mogu se očitovati loše dizajnirane sastavnice pa tako za naknadno dodavanje jednog stupca u bazu podataka, odnosno dodatnog tipa podatka u modelu, može biti potrebno izmijeniti gotovo svaku datoteku unutar projekta. Jednako tako, pri razvoju rješenja, bitno je iskoristiti ideje drugih ljudi kako bi se pružila dodatna dimenzija razmišljanja o problemu. Te ideje mogu biti neke općepoznate ideje o obrascima u programiranju ili pak savjet suradnika. Pri izgradnji rješenja, potrebno je biti otvorenog uma i prihvaćati različite perspektive. Sama aplikacija uspješno provodi zadane zadatke i ispunjava minimalne zahtjeve funkcionalnosti te se time može nazvati cjelovitom.

LITERATURA

[1] Orderman connection, [online], SCHULTES Microcomputervertriebs GmbH & Co. KG, dostupno na: https://schultes-kassen.de/wp-content/uploads/kopfbild_orderman7_5.png [14. rujna 2023.]

[2] The Orderman system, [online], Orderman GmbH, dostupno na: https://www.orderman.com/en/the-orderman-system [13. rujna 2023.]

[3] Get to Know TouchBistro's Restaurant POS System, [online], TouchBistro, dostupno na: https://www.touchbistro.com/blog/product-guide [5. rujna 2023.].

[4] TouchBistro product guide, [online], TouchBistro, dostupno na: https://www.touchbistro.com/wp-content/uploads/2022/09/homepage-carousel-order-mobile-1.jpg [5. rujna 2023.]

[5] Revel Systems, [online], Revel Systems, Inc, dostupno na: https://revelsystems.com [10. rujna 2023.]

[6] Lunchbox: fast online ordering that lets your brand shine, [online], Lunchbox Technologies, dostupno na: https://lunchbox.io/ordering [14. rujna 2023.]

[7] Goodtill: Goodeats, [online], Goodtill, dostupno na: https://thegoodtill.com/wp-content/uploads/2021/03/goodeats-header-image.png [14. rujna 2023.]

[8] Kotlin for Android, [online], Kotlin Foundation, dostupno na: https://kotlinlang.org/docs/android-overview.html [3. rujna 2023.]

[9] Jetpack Compose UI App Development Toolkit, [online], Google, dostupno na: https://developer.android.com/jetpack/compose [6. rujna 2023.]

[10] Meet Android Studio, [online], Google, dostupno na: https://developer.android.com/studio/intro [12. rujna 2023.]

[11] Cloud Firestore, [online], Google, dostupno na: https://firebase.google.com/docs/firestore [10. rujna 2023.]

[12] Room database diagram, [online], Google, dostupno na https://developer.android.com/static/images/training/data-storage/room_architecture.png [15. rujna 2023.]

[13] Save data in a local database using Room, [online], Google, dostupno na: https://developer.android.com/training/data-storage/room [13. rujna 2023.]

[14] Koin dependency injection, [online], Kotzilla, dostupno na: https://insert-koin.io [13. rujna 2023.]

[15] MVVM diagram, [online], GeeksforGeeks, dostupno na: https://media.geeksforgeeks.org/wp-content/uploads/20201002215007/MVVMSchema.png [15. rujna 2023.]

[16] R. Mishra, MVVM (Model View ViewModel) Architecture Pattern in Android, [online], GeeksforGeeks, dostupno na: https://www.geeksforgeeks.org/mvvm-model-view-viewmodel-architecture-pattern-in-android [13. rujna 2023.]

[17] M. Gargenta, Learning Android, O'Reilly Media, Inc., Sebastopol, 2011

[18] E. Hellman, Android Programming: Pushing the Limits, Packt Publishing Ltd.,

Birmingham, 2013.

SAŽETAK

Ovaj završni rad prelazi preko teme procesa izgradnje aplikacije dizajnirajući arhitekturu i implementirajući taj dizajn. Dizajn se ugrubo sastoji od opisa podatkovnog, aplikacijskog i prezentacijskog sloja. Prezentacijski sloj bavi se skladištenjem i dohvaćanjem podataka. Podatci se skladište u lokalnoj ili mrežnoj bazi podataka. Repozitoriji su ključ strukturiranog upravljanja skladištenjem i dohvaćanjem podataka iz baza podataka i njihove implementacije su jednostavne. Aplikacijski sloj rukuje tim podatcima. Rukovanje može biti jednostavno poput implementiranja metoda repozitorija koje skladište ili dohvaćaju podatke, ali može biti i puno složenije za teže zadatke. Prezentacijski sloj uključuje ekrane, komponente, stanja prikaza i modele prikaza kao i njihove funkcionalnosti. Na kraju, navigacijski sloj proširuje korisničko iskustvo kroz učinkovito rukovanje prijelazima među ekranima. U završnom radu je objašnjeno rukovanje iznimkama, serijalizacija podataka i više. Naime, glavna bit koja je predstavljena jest da sa čistim i strukturiranim pristupom dizajniranju rješenja, proces implementiranja dizajna zajedno sa odličnom skalabilnošću i održivošću je za veliku mjeru lakše.

Ključne riječi: Android, dizajn arhitekture, Kotlin, narudžba

ABSTRACT

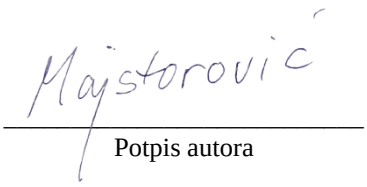
Title: **Application for managing orders in restaurants**

This final paper goes over the topic of the process of building an application by designing the architecture and implementing the design. The design roughly consists of the description of data, application and the presentation layers. The data layer handles the storage and retrieval of data. Data is stored in a local and or a network database. Repositories are the key to structurally manage the storage and retrieval of the data from the databases and they are simple in their implementations. The application layer handles the manipulation of the said data. The manipulation can be as simple as implementing repository methods which store and retrieve data, but it can also be quite more complex for more difficult tasks. The presentation layer encompasses the screens, components, viewstates and viewmodels and their functionalities. Finally, the navigation layer widens user experience through effectively handling the transitions between screens. The paper also explains exception handling, serialization of data and more. Though, the main point that it presents is that with clean and structured approach to software designing, the process of implementing the design along with having great scalability and maintainability is so much easier.

Keywords: Android, architecture design, Kotlin, order

ŽIVOTOPIS

Luka Majstorović rođen je u Vinkovcima 10. rujna 2001. godine. Osnovno obrazovanje započinje 2008. godine u Osnovnoj školi Siniše Glavaševića u Vukovaru. Nakon stjecanja osnovnog obrazovanja 2016. godine, započinje školovanje u Isusovačkoj klasičnoj gimnaziji s pravom javnosti u Osijeku. Završetkom srednje škole 2020. godine upisuje preddiplomski smjer računarstva, smjer programsko inženjerstvo na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku. U lipnju 2023. godine, započinje studentski posao u Razvojnom centru Osijek Financijske agencije na poziciji backend web programera.



Potpis autora

PRILOZI

Poveznica na GitHub repozitorij: https://github.com/lukamajstorovic/order-manager