Praćenje stanja zaliha namirnica

Čičak, Marin

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1011

PRAĆENJE STANJA ZALIHA NAMIRNICA

Marin Čičak

Zagreb, lipanj 2023.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1011

PRAĆENJE STANJA ZALIHA NAMIRNICA

Marin Čičak

Zagreb, lipanj 2023.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 10. ožujka 2023.

ZAVRŠNI ZADATAK br. 1011

Pristupnik:	**Marin Čičak (0036532647)**
Studij:	Elektrotehnika i informacijska tehnologija i Računarstvo
Modul:	Računarstvo
Mentor:	prof. dr. sc. Vedran Mornar

Zadatak: **Praćenje stanja zaliha namirnica**

Opis zadatka:

Načiniti mobilnu aplikaciju za praćenje stanja namirnica za operacijski sustav Android. Korisnik aplikacije treba imati mogućnost uređivanja popisa i karakteristika uređaja ili spremnika za pohranu namirnica, uređivanja kataloga namirnica s osnovnim atributima, te uređivanja sadržaja uređaja ili spremnika. Potrebno je izraditi izvješća o stanju namirnica po uređajima i po namirnicama te omogućiti slanje obavijesti korisniku u slučaju skorog isteka roka trajanja neke namirnice. Koristiti programske okvire Ionic, Angular te Spring Boot.

Rok za predaju rada: 9. lipnja 2023.

Sadržaj

Uvod

Ljudi se često nalaze u situaciji da pri kupnji namirnica nemaju popis potrebnih namirnica. Popisi namirnica pisani na papiru lako se izgube ili su nepregledni. Pamtiti spomenuti popis ne predstavlja optimalnu opciju. Navedeni problem potaknuo je razvoj ideje o kreiranju mobilne aplikacije.

Cilj ovog rada je kreirati mobilnu aplikaciju za operacijski sustav Android koja omogućuje praćenje stanja zaliha namirnica. Aplikacija pruža funkcionalnost praćenja stanja namirnica kreiranjem virtualnih uređaja koji čuvaju proizvode. Korisnik aplikacije može imati više virtualnih uređaja te ih redovito ažurirati. Korisnik je također ovlašten uređivati popis namirnica pomoću kojeg kreira proizvode. Pri svakom unosu novog proizvoda, bira odgovor na pitanje o primanju obavijesti o isteku roka upotrebe proizvoda. Dostupan je prikaz izvješća svih uređaja i njihovih sadržaja, kao i prikaz izvješća namirnica po trenutnima uređajima.

Podaci se spremaju u lokalnu bazu podataka na korisnikov mobilni uređaj. Za rad aplikacije nije potreban pristup internetu. Obavijesti o isteku roka upotrebe proizvoda šalju se dan prije upisanog isteka roka upotrebe. Za izradu aplikacije korišten je programski jezik *Kotlin* te alati *Jetpack Compose* i *Material Design*.

U sljedećim poglavljima opisane su funkcionalnosti koje pruža aplikacija. Navedene su i ukratko objašnjene tehnologije koje su korištene pri izradi aplikacije. Prikazana je arhitektura projekta uz opise pojedinih ključnih funkcionalnosti. Predočeni su neki od mogućih prikaza zaslona u aplikaciji prilikom njenog korištenja. Rad je uokviren zaključkom koji sadrži pregled svega istraženog, implementiranog i naučenog.

1. Zahtjevi

1.1. Funkcionalni zahtjevi

Funkcionalni zahtjevi opisuju kako bi se sustav trebao ponašati (koje usluge će pružati, kako će se ponašati u određenim situacijama, kako će reagirati na ulazne podražaje).

Implementirani su sljedeći zahtjevi:

- pregled trenutno postojećih uređaja u sustavu
- kreiranje novih uređaja
- uređivanje svojstava postojećih uređaja
- brisanje postojećih uređaja
- pregled postojećih proizvoda po uređaju
- kreiranje novih proizvoda za odabrani postojeći uređaj
- uređivanje svojstava postojećih proizvoda
- brisanje postojećih proizvoda
- pregled trenutne liste postojećih namirnica
- kreiranje novih namirnica
- uređivanje svojstava postojećih namirnica
- brisanje postojećih namirnica
- prikaz izvješća svih proizvoda po uređajima u kojima se nalaze
- prikaz izvješća svih uređaja s njihovim proizvodima
- slanje obavijesti o isteku roka upotrebe proizvoda

1.2. Ostali zahtjevi

- aplikacija mora podržavati hrvatsku abecedu (prikaz podataka na sučelju i unos u bazu podataka)
- programsko rješenje mora omogućiti dodavanje novih funkcionalnosti (bez narušavanja već postojećih funkcionalnosti aplikacije)
- aplikacija mora biti implementirana tako da korisnik svojim akcijama ne može izazvati grešku u sustavu
- operacije nad bazom (dodavanje, čitanje, mijenjanje i brisanje) se moraju izvršiti u nekoliko sekundi

2. Korištene tehnologije i alati

2.1. Android Studio

Android Studio je službeno integrirano razvojno okruženje (*Integrated development environment*, IDE[1]) za razvoj Android aplikacija. Android Studio temeljen je na IntelliJ IDEA uređivaču kôda. Uz funkcionalnosti navedenog alata, Android Studio nudi dodatne korisne funkcionalnosti. Alatom se može kreirati novi projekt ili otvoriti postojeći.

Jedna od posebnijih funkcionalnosti je stvaranje virtualnog uređaja preko kojeg se može pokrenuti aplikacija. Prilikom kreiranja virtualnog uređaja, na početku je potrebno definirati kategoriju uređaja (mobitel, tablet, TV, desktop, *wear OS*[2], softver za automobil) pa potkategoriju odabranog uređaja. Sljedeći korak je biranje verzije Android sustava (*system image*). Nakon toga se nude dodatne postavke (naknadna promjena već navedenih svojstava, imenovanje uređaja). To je posljednji korak prije izrade uređaja.

Osim kreiranja virtualnog uređaja i pokretanja aplikacije na njemu, aplikacija se može pokrenuti i na osobnom uređaju korisnika. Postoje dva načina povezivanja osobnog uređaja korisnika. Prvi način predstavlja povezivanje direktnim spajanjem računala i osobnog uređaja preko USB-a. Pokretanjem aplikacije u alatu, instalirat će se aplikacija na uređaju. U alatu na računalu dostupno je analiziranje aplikacije, dok se na uređaju mogu testirati sve funkcionalnosti aplikacije. Drugi način povezivanja je preko Wi-Fi mreže. Računalo i uređaj moraju biti spojeni na istu mrežu te se povezuju skeniranjem QR kôda. Detaljnije upute nalaze se na stranici *Run apps on a hardware device*.

---

[1] Služi za razvoj softvera, izgradnju, testiranje te upravljanje paketima (modulima). Često nudi funkcionalnosti automatskog uređivanja programskog kôda, označavanja teksta, procesa prevođenja (*compile*) te procesa pronalaženja grešaka (*debug*).

[2] Inačica Android operacijskog sustava za pametne satove.

2.2. Kotlin

Kotlin je programski jezik najčešće korišten pri izradi Android aplikacija. Kotlin je strogo tipizirani programski jezik visoke razine i otvorenog kôda (*open-source*). Pojam „strogo tipizirani“ znači da pazi na definiranje i upravljanje tipom podataka. Jezici visoke razine lakši su za održavanje jer zahtijevaju manje kôda. Zbog toga su greške u kôdu manje vjerojatne u odnosu na jezike niske razine. Također, jezici visoke razine su razumljiviji jer nude apstraktnije metode koje u svojem imenu opisuju svoju funkcionalnost.

Sintaksom je vrlo sličan programskom jeziku Java. Nudi funkcionalnosti objektno-orijentiranog načina programiranja, kao i funkcijskog načina. Još jedna bitna značajka programskog jezika jest više-platformsko korištenje (*cross-platform*).

2.3. Android Jetpack

Android Jetpack je skup biblioteka za pomoć pri pisanju kôda. Prednost korištenja biblioteka je smanjivanje standardnog kôda. Biblioteke se koriste preko *androidx* prostora imena (*namespace*[3]). Mnoge biblioteke pružaju proširenja za programski jezik Kotlin (Android KTX). Proširenja imaju ključnu riječ *ktx* koja se dodaje kao sufiks imenu biblioteke.

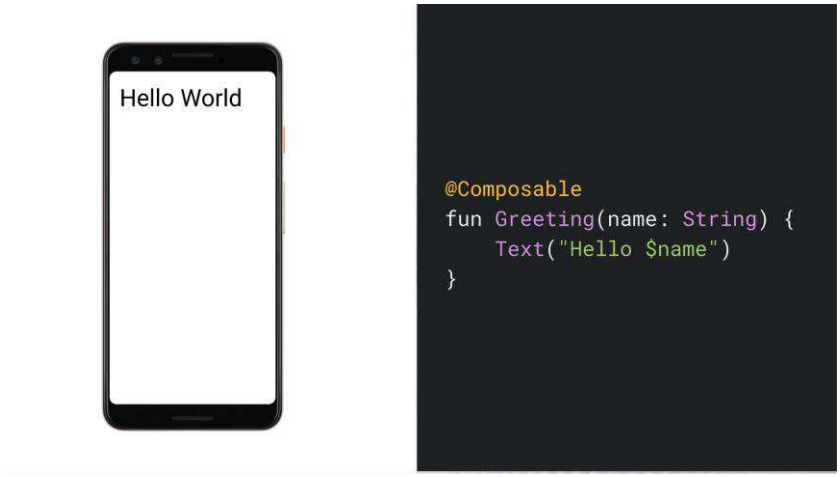
Neke od korištenih biblioteka prilikom izrade aplikacije su Room, Navigation i WorkManager. Room je biblioteka koja upravlja lokalnom bazom podataka koristeći SQLite[4]. Navigation je biblioteka koja pruža funkcionalnost kretanja kroz aplikaciju – navigacija po stranicama, odnosno komponentama aplikacije. WorkManager je biblioteka koja omogućuje pozadinski rad. Rad se može izvršavati prilikom rada aplikacije, ali i dok aplikacija nije aktivna.

[3] Prostor imena (*namespace*) osigurava razlikovanje objekata istog imena.

[4] Biblioteka relacijske baze podataka pisana u programskom jeziku C.

2.4. Jetpack Compose

Jetpack Compose je preporučeni alat za razvijanje korisničkog sučelja. Za razliku od standardnog razvijanja korisničkog sučelja aplikacije preko XML-a, Jetpack Compose je pisan u Kotlinu i zahtijeva puno manje kôda. Kôd je razumljiviji, lakši za testiranje i pronalaženje grešaka. Korisničko sučelje gradi se komponentama[5] koje se mogu ponovno koristiti na više različitih mjesta. Na slici (Slika 1) je prikazan kôd i korisničko sučelje jednostavne komponente.



Slika 1 Prikaz jednostavne komponente

2.5. Material Design

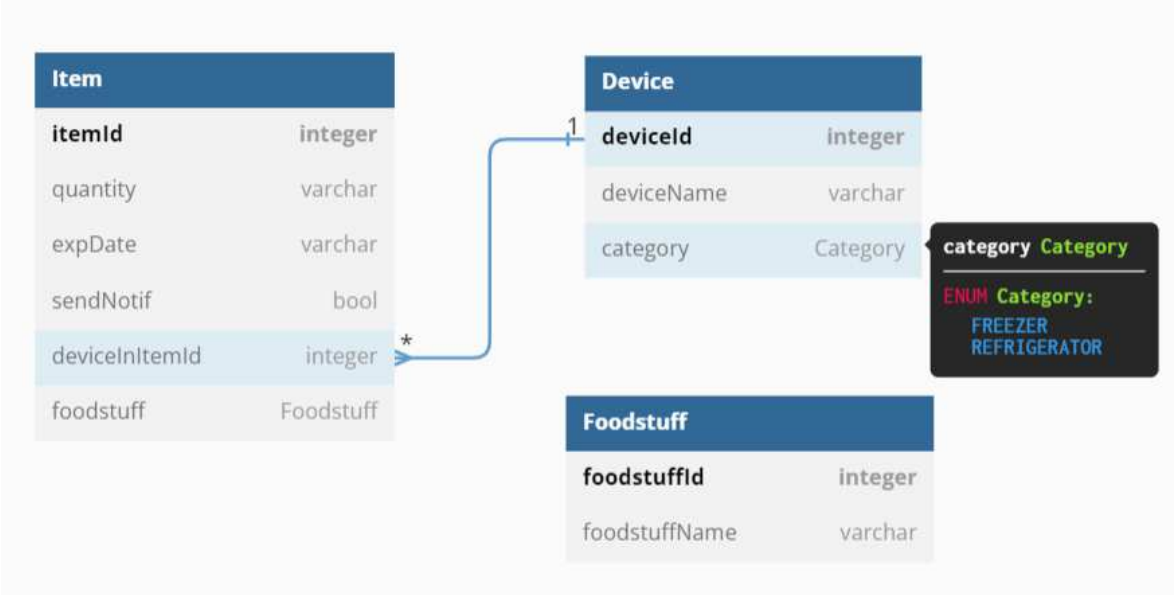
Material Design je Google-ov sustav za izradu dizajna. Pruža razne smjernice i alate za izradu korisničkog sučelja. Često se koristi kod izrade korisničkog sučelja Android aplikacija, no i u razvoju web aplikacija. Sustav nudi funkcionalnosti korištenja dostupnih ikona, oblika, stilova fonta te animacija. Također je moguće izraditi temu, odnosno paletu boja koja se koristi u izradi korisničkog sučelja (dostupno na poveznici). Trenutna najnovija verzija je Material 3.

[5] Kompozicijske funkcije (*Composable*) koje primaju podatke i prikazuju elemente korisničkog sučelja.

3. Arhitektura

3.1. Model baze podataka

U aplikaciji je korištena relacijska baza podataka. Baza se sastoji od 3 entiteta. To su *Item*, *Device* i *Foodstuff*. Dijagram baze podataka (Slika 2) je izrađen u alatu dbdiagram.io.



Slika 2 Model relacijske baze podataka

Tablica *Foodstuff* sadrži podatke o trenutno dostupnim namirnicama koje korisnik koristi za stvaranje proizvoda. Ima atribut *foodstuffName* koji se koristi kao ime namirnice, a samim time i kao ime proizvoda. Tablica *Item* sadrži podatke o svim proizvodima u aplikaciji. Povezana je s tablicom *Device* (preko atributa *deviceInItemId*) jer je svaki proizvod u točno jednom uređaju, dok jedan uređaj može imati nula ili više proizvoda. Atributi tablice *Item* su *quantity* koji označava količinu proizvoda, *expDate* koji označava datum isteka roka upotrebe namirnice, *sendNotif* koji govori o tome hoće li korisnik dobiti obavijest iz aplikacije ili ne. Tablica sadrži i atribut *foodstuff* koji može biti samo vrijednost iz tablice *Foodstuff*. Tablica *Device* opisuje uređaje koji se koriste u aplikaciji.

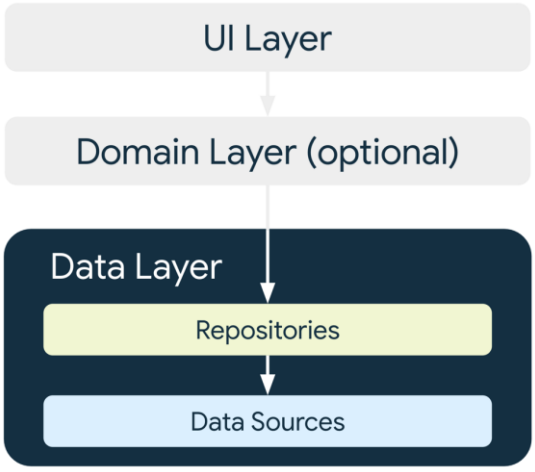
Atribut *category* je vrijednost iz enum[6] objekta, dok je *deviceName* ime uređaja. Svi entiteti imaju jedinstveni identifikator koji služi za njihovo razlikovanje.

3.2. Dizajn sustava

Arhitektura programskog rješenja temelji se na **model-pogled-pogled na model** (*model-view-viewmodel*, MVVM) arhitekturi (Slika 3). Slična je poznatoj MVC[7] arhitekturi, no razlika je u novom sloju *pogled na model*. Navedeni sloj olakšava prikaz podataka tako što ih oblikuje na prilagođen način za lakše upravljanje. Razdvajanje aplikacijske logike i korisničkog sučelja pomaže pri testiranju aplikacije, ponovnom korištenju kôda i dodavanju novih funkcionalnosti.

Pogled je zaslužan za izgled korisničkog sučelja. Najčešće nema logiku, no ako ima nije pretjerano složena. Pogled čuva stanja aplikacije te tako javlja promjene „pogledu na model“. To su većinom podaci koje korisnik unosi ili pritiskanje na gumb. Također, ovisno o stanju, pogled prikazuje (ili ne prikazuje) određene podatke ili komponente.

Pogled na model na sebi čuva svu logiku. Reagira na promjenu stanja u interakciji s pogledom. Bitna funkcija je dohvaćanje podataka s modela i slanje tih podataka pogledu. Podaci se često prije slanja prikladno oblikuju kako bi ih pogled lakše prikazao. Osim dohvaćanja podataka, pogled na model šalje podatke (također je poželjno oblikovanje podataka te njihova provjera, odnosno validacija) koje korisnik sprema u bazu.



Slika 3 Prikaz dizajna arhitekture

Model oblikuje bazu podataka i ima direktnu komunikaciju s bazom. Prima zahtjeve od pogleda na model te ovisno o zahtjevu komunicira s bazom podataka. Model oblikuje bazu tako što definira njene entitete i pripadna svojstva entiteta, kao i veze među entitetima.

[6] Poseban tip podataka koji se koristi za spremanje konstanti, odnosno nepromjenjivih vrijednosti.

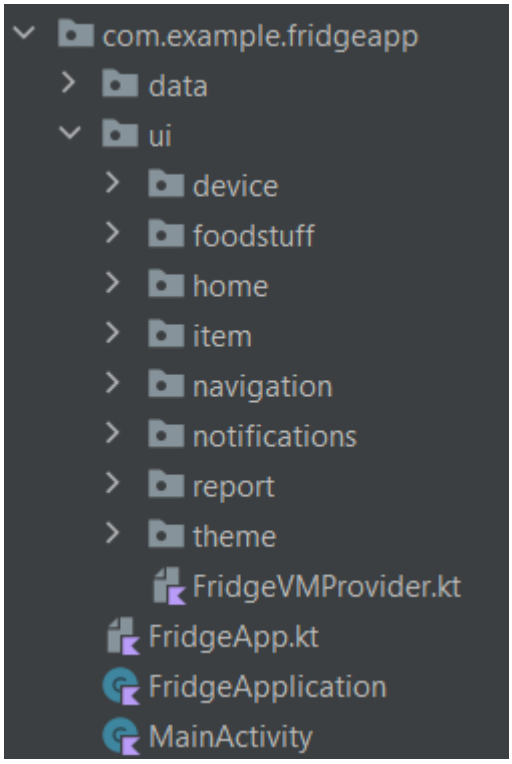
[7] Model-pogled-nadglednik (*model-view-controller*) arhitektura.

U aplikaciji postoji još jedan sloj, između modela i pogleda na model (*repository*). To su funkcije koje poziva pogled na model, a one dalje pozivaju funkcije modela. Ovo je dobra praksa jer nekad nije poželjno direktno koristiti podatke iz baze, nego ih po potrebi filtrirati.

Podaci se često mijenjaju stoga se kod prikaza na korisničkom sučelju moraju ažurirati. Kako se ne bi stalno provjeravalo jesu li podaci promijenjeni, tip podatka *Flow* (često se uspoređuje sa strujom, *stream*) se pobrinuo za to. Dohvaćanje *Flow* podataka odvija se u asinkronim komponentama (*coroutines*) kako se ne bi blokirala glavna dretva. Ovisno o promjeni podataka u bazi podataka, podaci dohvaćeni u tipu *Flow* će se automatski ažurirati. Nakon dohvaćanja podataka iz baze, podatke tipa *Flow* poželjno je pretvoriti u tip *StateFlow*. Tip *StateFlow* pruža svojstvo čuvanja podataka na kratki period nakon prestanka dohvaćanja istih. Na primjer, prilikom okretanja zaslona iz okomitog u vodoravni položaj, korisničko sučelje se mora ponovno rekonstruirati (stranica se mora ponovno učitati). Prilikom ponovnog učitavanja stranice, podaci se moraju dohvatiti iz baze podataka i prikazati. Zbog navedenog svojstva o čuvanju podataka na kratki period, korištenjem *StateFlow* objekta za prikaz i dohvaćanje podataka nije ih potrebno ponovno dohvaćati. Bitna je još jedna karakteristika ovakvog pristupa u dohvaćanju i pohrani podataka. Prilikom prelaska na novu stranicu, memorija za pohranu podataka koji su se dohvaćali za prethodnu stranicu se oslobađa jer trenutno nisu potrebni. Time se resursi uređaja i aplikacije ne troše nepotrebno. Tako se osigurava optimalnost rada aplikacije.

3.3. Arhitektura programskog rješenja

Projekt je napravljen u alatu *Android Studio*. Pokretanjem alata, klikom na gumb *New,* otvara se prozor s već gotovim uzorcima korisničkog sučelja. Tu su ponuđene već spomenute kategorije (mobitel, tablet, *wear OS*…) za odabir. Aplikacija je kreirana za mobitel te je korišten uzorak *Empty activity*. Nakon toga bira se ime projekta i lokacija gdje će se spremiti projekt te minimalna verzija Android sustava. Projekt zatim mora preuzeti sve potrebne ovisnosti i izgraditi aplikaciju, što može potrajati neko vrijeme. Nakon svih koraka projekt je spreman za rad.



Slika 4 Prikaz paketa projekta

Projekt je složen po paketima (direktorijima). Početni paket projekta je *com.example.fridgeapp*. U njemu se nalaze dva poddirektorija: *data* u kojemu su datoteke vezane za komunikaciju s bazom (model) i *ui* u kojemu su paketi vezani za korisničko sučelje. Osim navedenih paketa nalazi se datoteka *MainActivity.kt* koja je početna datoteka te se ona prva poziva pri svakom pokretanju aplikacije. *FridgeApp.kt* sadrži funkciju (*@Composable*) koja je pozvana iz početne datoteke te ona prikazuje aplikaciju. Uz navedenu metodu, sadrži komponentu koja se koristi na svakoj stranici, a prikazuje gornju traku (ovisno o stranici, gornja traka sadrži gumbe za povratak, uređivanje i brisanje trenutnog sadržaja). Datoteka unutar *ui* paketa, *FridgeVMProvider.kt*, u sebi ima varijablu koja inicijalizira sve potrebne poglede na model koji se koriste u aplikaciji.

Paket *ui* ima u sebi pakete vezane za svaki entitet baze, kao i paket za početnu stranicu, izvješća, navigaciju i obavijesti. Paketi *device*, *item* i *foodstuff* sadrže datoteke ovisne o entitetima po kojima su nazvane. Pregled paketa dan je u nastavku:

- Paket *theme* je automatski generirani paket pri izradi projekta te sadrži datoteke koje opisuju boje, stilove za tekst i temu za cijelu aplikaciju. Za projekt je kreirana nova paleta boja u sustavu Material Design te je implementirana na potrebnim mjestima.
- Paket *navigation* ima u sebi dvije datoteke. Jedna datoteka je objekt, koji ima varijable *route* (dio putanje prilikom navigacije) i *pageTitle* (označava naslov na pojedinoj stranici), koji se implementira u svakoj stranici. Druga datoteka sadrži sve potrebne navigacije (kretnje kroz aplikaciju) te prikazuje određenu stranicu ovisno o trenutnoj ruti. Osim prikazivanja stranice, pamti poslane parametre (najčešće je identifikator koji se koristi na odabranoj stranici) i prikazuje, odnosno

ne prikazuje određene komponente (pruža, odnosno ne pruža funkcionalnosti uz komponente).

- Paket *home* prikazuje početnu stranicu – u ovome slučaju trenutne postojeće uređaje. Ima dvije datoteke, pogled i pogled na model. Pogled na model dohvaća trenutne uređaje te ih šalje pogledu, koji ih prikazuje. Izgled je napravljen pomoću nekoliko komponenti. Ako nema niti jednog uređaja u bazi, prikazat će se pripadna poruka. Klikom na uređaj, odlazi se na novu stranicu gdje su prikazani proizvodi za taj uređaj. Na stranici, dolje u desnom kutu, nalazi se gumb koji vodi na formu gdje se kreira novi uređaj. Novi uređaj se kreira definiranjem njegovom imena i kategorije. Na gornjoj traci u desnom kutu, nalazi se gumb koji vodi na stranicu za izvještaje.
- Paket *report* sadrži datoteke koje prikazuju izvještaj. Početna stranica prikazuje dva gumba, koji na pritisak vode na određenu stranicu. Gumbi vode na stranice vezane za izvještaj po uređajima ili po namirnicama. Izvještaj po uređajima prikazuje sve trenutne uređaje i njihove sadržaje. Izvještaj po namirnicama prikazuje skočni izbornik iz kojeg se bira namirnica. Odabirom namirnice, prikazuju se svi uređaji te atributi odabrane namirnice u uređajima. Osim prikaza, stranice ne pružaju nikakve druge funkcionalnosti.
- Paket *device* sadrži datoteke koje prikazuju sadržaj uređaja (proizvodi unutar uređaja te njihova svojstva) te formu gdje se kreira i uređuje uređaj. Ovisno o pozivu kreiranja ili uređivanja uređaja, forma će biti prazna ili će sadržavati podatke o određenome uređaju. Klik na gumb za uređivanje, preusmjerava na formu za kreiranje uređaja i forma sadrži podatke za navedeni uređaj. Klikom na gumb za brisanje, otvara se skočni prozor koji traži za dopuštenje za brisanjem. Datoteke pogleda na model sadrže pripadne funkcije za dohvaćanje i slanje podataka, kao i funkcije za pretvorbu podataka radi lakšeg upravljanja.
- Paket *item* sadrži datoteke koje prikazuju formu za kreiranje ili uređivanje proizvoda. Klikom na gumb za kreiranje, otvara se forma za kreiranje proizvoda. Podatak za ime proizvoda bira se iz skočnog izbornika, koji ima listu namirnica (*Foodstuff*). Unos svih podataka je obavezan. Klik na proizvod iz liste proizvoda u uređaju, preusmjerava na formu koja je popunjena podacima o tom proizvodu. Odlaskom na formu kod uređivanja proizvoda, na gornjoj traci dostupan je gumb za

brisanje koji nudi mogućnost brisanja proizvoda. Kao i kod uređaja, pogledi na model imaju pripadne funkcije za dohvat i slanje podataka, te funkcije za pretvaranje tipa podataka.

- Paket *foodstuff* ima datoteke slične datotekama za uređaj. Nudi se prikaz liste namirnica te gumb za kreiranje nove namirnice. Klik na gumb za kreiranje namirnice, preusmjerava na formu gdje se upisuje ime nove namirnice. Klikom na namirnicu iz liste namirnica, preusmjerava se na formu gdje je upisano ime namirnice te se tamo može promijeniti. Prilikom izmjene podataka, na gornjoj traci je dostupan gumb za brisanje.
- Paket *notifications* sadrži dvije datoteke koje služe pri slanju obavijesti. Jedna datoteka sadrži funkciju za stvaranje i slanje obavijesti (naslov, poruka, ikona), funkciju koja računa vrijeme kada treba poslati obavijest (dan prije unesenog datuma + 15 sekundi za malu razliku) te funkciju koja nudi mogućnosti ulaska u aplikaciju prilikom pritiskanja obavijesti. Druga datoteka je klasa koja nasljeđuje klasu CoroutineWorker[8]. Iz nje se poziva navedena funkcija za kreiranje i slanje obavijesti.

Paket *data* sadrži datoteku u kojoj su definirani entiteti, datoteku gdje su definirani upiti nad bazom (*data access objects*, DAO – sučelje), datoteke koje pozivaju upite nad bazom (sloj između modela i pogleda na model, *repository*) te datoteke koje inicijaliziraju bazu. Room biblioteka kreira relacije nad entitetima pomoću novih objekata (podatkovnih klasa) gdje se moraju definirati primarni i strani ključ. Anotacija za relaciju je *@Relation* koja prima parametre ključeva. Također, anotacijom *@Embedded* definira se entitet koji se referencira u upitima nad bazom. U primjeru se vidi podatkovna klasa (Slika 5) koja ima uređaj i listu proizvoda. Anotacijom *@Embedded* nad uređajem,

```
data class DeviceWithItems(
    @Embedded val device: Device,
    @Relation(
        parentColumn = "deviceId",
        entityColumn = "deviceInItemId"
    )
    val items: List<Item>
)
```

Slika 5 Podatkovna klasa s relacijama

[8] Klasa koja izvršava asinkroni rad u pozadini. Potrebno je prilikom nasljeđivanja ove klase implementirati metodu doWork() koja se poziva te odrađuje željeni rad. Ako se rad ne izvrši unutar 10 minuta, poslat će se signal za zaustavljanje.

tako se može pozvati upit nad bazom za entitet uređaj. Prilikom izvršavanja upita (Slika 6), definiranjem povratnog tipa podatka kao nove podatkovne klase, Room zna da treba vratiti takav objekt, a ne samo podatke o uređaju (što bi uobičajeni upit vratio). Upit ima anotaciju *@Transaction* zato što je definiran povratni tip nove podatkovne klase, stoga treba izvršiti dva upita.

```
@Transaction
@Query("SELECT * FROM device WHERE deviceId = :id")
fun getDeviceItems(id: Int): Flow<DeviceWithItems>
```

Slika 6 Funkcija za pozivanje upita nad bazom

Bitne su još dvije funkcije za rad s obavijestima (Slika 7). Prva je za slanje obavijesti. Prima parametre za stvaranje poruke (ime proizvoda kojem istječe rok i ime uređaja u kojem se nalazi taj proizvod), identifikatora te perioda vremena (za koliko se treba poslati obavijest). Podaci se šalju kroz objekt klase Data, koja služi za prijenos podataka vezanih za Worker klase (u ovome slučaju, CoroutineWorker klasa). Stvara se instanca za rad koji se ne ponavlja (OneTimeWorkRequest, nasljeđuje apstraktnu klasu WorkRequest). Preko instance klase WorkManager[9], kreira se jedinstveni rad koji se ne ponavlja. Rad prima kao parametre jedinstveni identifikator, enum ExistingWorkPolicy[10] i instancu za rad koji se ne ponavlja. Druga funkcija prekida rad, odnosno prekida slanje obavijesti s navedenim identifikatorom.

[9] Klasa koja definira rad koji se treba izvršiti. Rad se može pratiti te se mogu definirati zahtjevi o kojima ovisi izvršavanje rada.

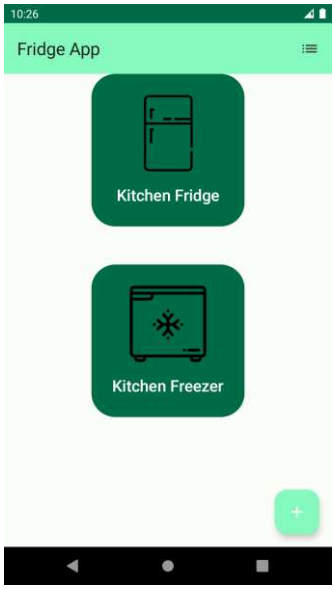
[10] Enum ExistingWorkPolicy sadržava konstante vezane za konflikte u jedinstvenom radu koji se ne ponavlja.

```kotlin
private val workManager = WorkManager.getInstance(context)

override fun scheduleNotification(
    duration: Long,
    unit: TimeUnit,
    itemName: String,
    deviceName: String,
    id: Int
) {
    val data = Data.Builder()
    data.putString(NotificationWorker.itemId, id.toString())
    data.putString(NotificationWorker.itemName, itemName)
    data.putString(NotificationWorker.deviceName, deviceName)

    val workRequestBuilder = OneTimeWorkRequestBuilder<NotificationWorker>()
        .setInitialDelay(duration, unit)
        .setInputData(data.build())
        .build()

    workManager.enqueueUniqueWork(
        id.toString(),
        ExistingWorkPolicy.REPLACE,
        workRequestBuilder
    )

}

override fun cancelNotification(id: Int) {
    workManager.cancelUniqueWork(id.toString())
}
```

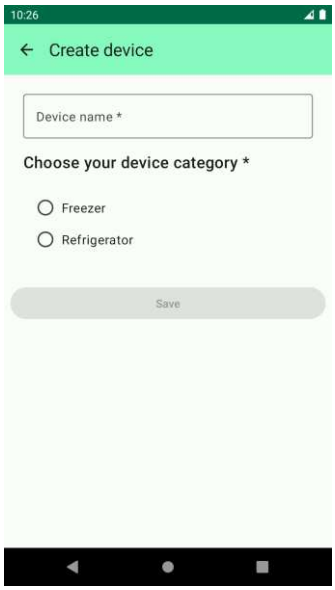
Slika 7 Funkcije za rad s obavijestima

4. Prikaz

U ovome poglavlju prikazani su neki od mogućih prikaza aplikacije.

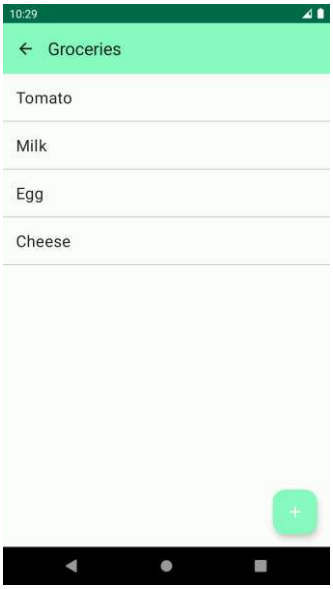


Slika 8 Početna stranica

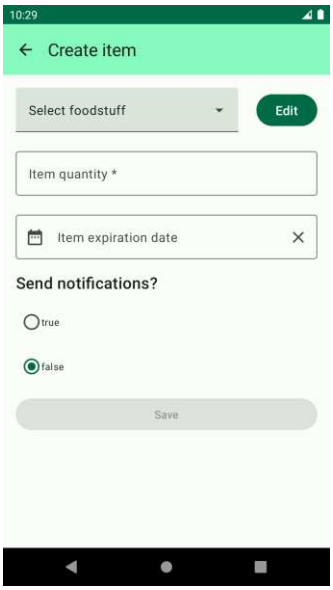


Slika 9 Kreiranje uređaja

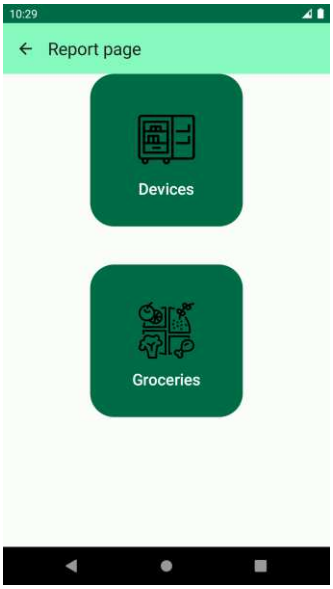
Slika 10 Sadržaj uređaja



Slika 11 Lista namirnica



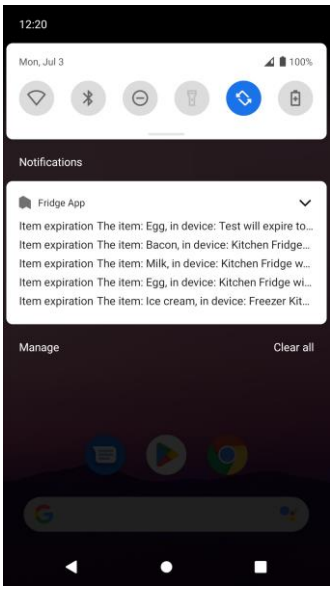
Slika 12 Unos proizvoda



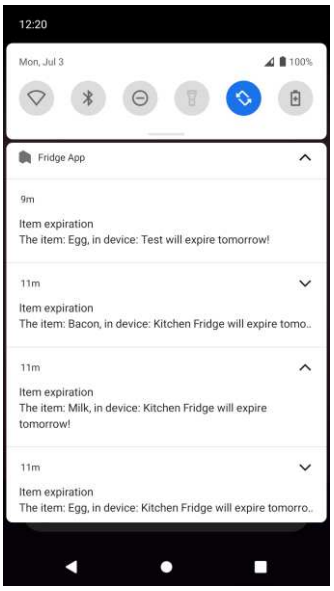
Slika 13 Izvještaji

Slika 14 Izvještaj po namirnicama

Slika 15 Izvještaj po uređajima



Slika 16 Skraćene obavijesti



Slika 17 Proširene obavijesti

Zaključak

Prije izrade samog projekta, važno je dobro definirati funkcionalne zahtjeve i arhitekturu koja će biti najprikladnija za njegovu izradu. Nakon definiranja zahtjeva, vrlo je važno dobro definirati model baze podataka. Detaljnijim i podrobnijim proučavanjem tehnologija koje se koriste u njegovoj izradi, osigurava se brži i efikasniji rad na projektu. To uključuje praćenje aktualnih verzija alata, praćenje najnovijih verzija programskih jezika, čitanje službene dokumentacije i službenih video-materijala.

Iz potrebe za praćenjem vlastitih zaliha namirnica razvijena je aplikacija koja to omogućuje. Korisnik aplikacije može neovisno o internetskoj vezi pristupati aplikaciji te ju sukladno njenim funkcionalnostima koristiti. Aplikacija nudi funkcionalnosti pregledavanja sadržaja te kreiranja novog ili uređivanja postojećeg.

Izrađena aplikacija omogućuje nadogradnju novih funkcionalnosti bez ometanja već postojećih. Uz nadogradnju funkcionalnosti, moguća je nadogradnja baze podataka kao i spremanje baze na poslužitelj. Time se omogućuje sinkronizacija jedne baze između više korisnika, ali i sinkronizacija jednog korisnika putem više uređaja. Također, aplikacija je mobilna te se može nadograditi za *wear OS*.

Literatura

Alan Jović, N. F. (Lipanj 2023). *Fakultet elektrotehnike i računarstva*. Dohvaćeno iz Procesi programskog inženjerstva: https://www.fer.unizg.hr/_download/repository/Procesi_programskog_inzenjerstva_3_izdanje.pdf

Atitienei, D. (Lipanj 2023). *Medium*. Dohvaćeno iz Date and Time pickers in Jetpack Compose: https://medium.com/@daniel.atitienei/date-and-time-pickers-in-jetpack-compose-f641b1d72dd5

Barr, J. (Lipanj 2023). *Amazon Web Services*. Dohvaćeno iz What Is An IDE (Integrated Development Environment)?: https://aws.amazon.com/what-is/ide/

Becris. (Lipanj 2023). *Flaticon*. Dohvaćeno iz Nutrition - Free technology icons: https://www.flaticon.com/free-icon/nutrition_649371

Freepik. (Lipanj 2023). *Flaticon*. Dohvaćeno iz Fridge - Free technology icons: https://www.flaticon.com/free-icon/fridge_534182

Freepik. (Lipanj 2023). *Flaticon*. Dohvaćeno iz Fridge - Free technology icons: https://www.flaticon.com/free-icon/fridge_229632

Google. (Lipanj 2023). *Android Developers*. Dohvaćeno iz androidx.compose-material: https://developer.android.com/reference/kotlin/androidx/compose/material/package-summary#radiobutton

Google. (Lipanj 2023). *Android Developers*. Dohvaćeno iz Android KTX | Kotlin: https://developer.android.com/kotlin/ktx

Google. (Lipanj 2023). *Android Developers*. Dohvaćeno iz Meet Android Studio: https://developer.android.com/studio/intro

Google. (Lipanj 2023). *Android Developers*. Dohvaćeno iz Create and manage virtual devices: https://developer.android.com/studio/run/managing-avds

Google. (Lipanj 2023). *Android Developers*. Dohvaćeno iz Thinking in Compose | Jetpack Compose: https://developer.android.com/static/images/jetpack/compose/mmodel-simple.png

Google. (Lipanj 2023). *Android Developers*. Dohvaćeno iz Data Layer: https://developer.android.com/static/codelabs/basic-android-kotlin-compose-persisting-data-room/img/991d77e62a5f71e8_960.png

Google. (Lipanj 2023). *Android Developers*. Dohvaćeno iz Android Basics with Compose course: https://developer.android.com/courses/android-basics-compose/course

Google. (Lipanj 2023). *Android Developers*. Dohvaćeno iz CoroutineWorker: https://developer.android.com/reference/kotlin/androidx/work/CoroutineWorker

Google. (Lipanj 2023). *Android Developers*. Dohvaćeno iz Create and manage notification channels: https://developer.android.com/develop/ui/views/notifications/channels

Google. (Lipanj 2023). *Android Developers*. Dohvaćeno iz Create a notification: https://developer.android.com/develop/ui/views/notifications/build-notification

Google. (Lipanj 2023). *Android Developers*. Dohvaćeno iz Create a group of notifications: https://developer.android.com/develop/ui/views/notifications/group

Google. (Lipanj 2023). *Android Developers*. Dohvaćeno iz WorkManager: https://developer.android.com/reference/androidx/work/WorkManager

Google. (Lipanj 2023). *Android Developers*. Dohvaćeno iz OneTimeWorkRequest: https://developer.android.com/reference/androidx/work/OneTimeWorkRequest

Google. (Lipanj 2023). *Android Developers*. Dohvaćeno iz WorkRequest: https://developer.android.com/reference/androidx/work/WorkRequest

Google. (Lipanj 2023). *Android Developers*. Dohvaćeno iz ExistingWorkPolicy: https://developer.android.com/reference/androidx/work/ExistingWorkPolicy

Google. (Lipanj 2023). *Android Developers*. Dohvaćeno iz Define relationship between objects: https://developer.android.com/training/data-storage/room/relationships

Google. (Lipanj 2023). *Android Developers*. Dohvaćeno iz Getting started with Android Jetpack: https://developer.android.com/jetpack/getting-started

Smartline. (Lipanj 2023). *Flaticon*. Dohvaćeno iz Fridge - Free technology icons: https://www.flaticon.com/free-icon/fridge_857336

Stonis, M. (Lipanj 2023). *Microsoft Learn*. Dohvaćeno iz Model-View-ViewModel: https://learn.microsoft.com/en-us/dotnet/architecture/maui/mvvm

vectorsmarket15. (Lipanj 2023). *Flaticon*. Dohvaćeno iz Fridge - Free technology icons: https://www.flaticon.com/free-icon/fridge_2645417

Zhukovich, A. (Lipanj 2023). *Mobile development & testing with Alex*. Dohvaćeno iz Jetpack Compose: Dropdown Menu: https://alexzh.com/jetpack-compose-dropdownmenu/

Sažetak

Praćenje stanja zaliha namirnica

Aplikacija je kreirana za operacijski sustav Android, a kôd je pisan programskim jezikom Kotlin. Pri izradi se koristio uređivač koda Android Studio te alat Jetpack Compose. Aplikacija omogućuje izradu i uređivanje virtualnih uređaja za spremanje proizvoda. U uređajima je moguće spremanje proizvoda koji se kreiraju pomoću popisa namirnica. Prilikom kreiranja proizvoda, korisnik bira želi li primati obavijesti o isteku roka upotrebe proizvoda. Korisnik može pregledavati i uređivati postojeće proizvode uređaja. Aplikacija pruža funkcionalnosti pregledavanja i uređivanja kataloga namirnica. Dostupan je pregled izvješća o uređajima i namirnicama. Aplikacija ne zahtijeva internetsku vezu za korištenje. Podaci se spremaju na mobilni uređaj u lokalnu bazu podataka.

Android aplikacija, Kotlin, Android Studio, Android Jetpack, Jetpack Compose

Summary

Monitoring of the food inventory

The application was created for the Android operating system, and the code was written in the Kotlin programming language. The code editor Android Studio and the Jetpack Compose tool were used during the creation. The application enables the creation and editing of virtual devices for storing products. In the devices, it is possible to save products that are created using a grocery list. When creating a product, the user chooses whether he wants to receive notifications about the expiration of the product's expiration date. User can view and edit existing device products. The application provides functionality for viewing and editing the grocery catalog. A preview of device and grocery reports is available. The application does not require an internet connection to use. The data is saved on the mobile device in a local database.

Android application, Kotlin, Android Studio, Android Jetpack, Jetpack Compose