Android aplikacija za praćenje tijeka terapije

Suk, Sven

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

IZRADA ANDROID APLIKACIJE ZA PRAĆENJE TIJEKA TERAPIJE

Završni rad

Sven Suk

Osijek, 2020.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

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

Osijek, 01.09.2020.

Odboru za završne i diplomske ispite

Prijedlog ocjene završnog rada na preddiplomskom sveučilišnom studiju

Ime i prezime studenta:	Sven Suk
Studij, smjer:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	R4140, 19.09.2019.
OIB studenta:	47891141842
Mentor:	Doc.dr.sc. Mirko Köhler
Sumentor:	
Sumentor iz tvrtke:	
Naslov završnog rada:	Android aplikacija za praćenje tijeka terapije
Znanstvena grana rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Predložena ocjena završnog rada:	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 2 bod/boda Jasnoća pismenog izražavanja: 2 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene mentora:	01.09.2020.
Datum potvrde ocjene Odbora:	09.09.2020.

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

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

IZJAVA O ORIGINALNOSTI RADA

Osijek, 10.09.2020.

Ime i prezime studenta:	Sven Suk
Studij:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	R4140, 19.09.2019.
Turnitin podudaranje [%]:	6

Ovom izjavom izjavljujem da je rad pod nazivom**: Android aplikacija za praćenje tijeka terapije**

izrađen pod vodstvom mentora Doc.dr.sc. Mirko Köhler

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:

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

IZJAVA

o odobrenju za pohranu i objavu ocjenskog rada

kojom ja __Sven Suk_____, OIB: __47891141842_____, student/ica Fakulteta elektrotehnike, računarstva i informacijskih tehnologija Osijek na studiju _Preddiplomski sveučilišni studij Računarstvo _, kao autor/ica ocjenskog rada pod naslovom: _Android aplikacija za praćenje tijeka terapije**,**
dajem odobrenje da se, bez naknade, trajno pohrani moj ocjenski rad u javno dostupnom digitalnom repozitoriju ustanove Fakulteta elektrotehnike, računarstva i informacijskih tehnologija Osijek i Sveučilišta te u javnoj internetskoj bazi radova Nacionalne i sveučilišne knjižnice u Zagrebu, sukladno obvezi iz odredbe članka 83. stavka 11. *Zakona o znanstvenoj djelatnosti i visokom obrazovanju* (NN 123/03, 198/03, 105/04, 174/04, 02/07, 46/07, 45/09, 63/11, 94/13, 139/13, 101/14, 60/15).
Potvrđujem da je za pohranu dostavljena završna verzija obranjenog i dovršenog ocjenskog rada. Ovom izjavom, kao autor/ica ocjenskog rada dajem odobrenje i da se moj ocjenski rad, bez naknade, trajno javno objavi i besplatno učini dostupnim:

a) široj javnosti

b) studentima/icama i djelatnicima/ama ustanove

c) široj javnosti, ali nakon proteka 6 / 12 / 24 mjeseci (zaokružite odgovarajući broj mjeseci).

**U slučaju potrebe dodatnog ograničavanja pristupa Vašem ocjenskom radu, podnosi se obrazloženi zahtjev nadležnom tijelu Ustanove.*

__Osijek, 10.09.2020._____ _____

(mjesto i datum) (vlastoručni potpis studenta/ice)

SADRŽAJ

1. UVOD

Uzimanje lijekova pacijentima stvara određeni oblik obveze jer svaki dan u isto vrijeme trebaju popiti propisani lijek. Iz toga razloga neki pacijenti svoje podsjetnike pišu na papire, ali većina njih oslanja se ipak na svoje pamćenje, koje često zakaže. U Hrvatskoj je provedeno testiranje na uzorku od 100 ljudi iznad 40 godina starosti koji pripadaju skupini visokog rizika za kardiovaskularne bolesti. Rezultati ovoga testiranja su zabrinjavajući jer manje od polovice ispitanika pije lijekove redovito. Od njih 60 posto koji zaboravljaju popiti svoje dnevne terapije, 25 posto ispitanika to čini jedanput u tjedan dana, dok 8 posto redovno zaboravlja popiti propisani lijek. Svaki puta kada čovjek zaboravi popiti lijek on se dovodi u rizčnu situaciju. Kardiovaskularne bolesti trenutno su vodeći uzrok smrti i invaliditeta u svijetu. Svake godine od njih umre više od 17.5 milijuna ljudi.

Izrada mobilne aplikacije potaknuta je ovim rezultatima. Aplikacija se temelji na ideji da korisnicima omogući brzi unos lijeka skeniranjem QR koda. Unutar njega se nalaze podatci o lijeku i njegovim skeniranjem korisnik započinje uzimati terapiju. To znači da će se korisniku u ovisnosti o podatcima iz QR koda javljati pravovremene obavijesti koje predstavljaju podsjetnike na terapiju. Pomoću ove aplikacije korisnik se ne mora više oslanjati na svoje pamćenje i brinuti da će opet zaboraviti popiti propisani lijek. Aplikacija će se izrađivati u Android Studio, a programski jezik koji će se koristiti je Java. Stoga je potrebno imati osnovna predznanja o Javi kako bi se mogao razumijeti kod.

Glavni dio rada podijeljen je na tri dijela: usporedba s već postojećim rješenjima, razvojno okruženje i programsko rješenje zadatka. U prvom dijelu se uspoređuje aplikacija koja se izrađuje ovim završnim radom s već postojećim sličnim rješenjima, a to obuhvaća iznošenje prednosti i mana. U drugom dijelu opisano je razvojno okruženje koje je korišteno za izradu aplikacije. U zadnjem dijelu se detaljno prolazi kroz progmrasko rješenje zadatka.

1.1. Zadatak završnog rada

Cilj je izraditi Android aplikaciju koja će moći skenirati QR kod u kojemu će biti spremljene stavke naziv, količina i opis lijeka te trajanje terapije i ponavljanje u jednom danu. Te će se informacije spremati u aplikaciju i na osnovnu njih korisniku će se javljati alarmi prije uzimanja terapije. Nakon prestanka trajanja terapije ti podatci odlaze u korisnikovu medicinsku povijest kako bi korisnik imao predodžbu o svojim prijašnjim terapijama.

2. USPOREDBA S VEĆ POSTOJEĆIM RJEŠENJIMA

U današnje vrijeme sve je teže osmisliti nešto novo, a razlog tomu je što gotovo svi imaju pristup internetu. Gotovo svaka ideja koja nekome prođe kroz glavu je već realizirana, ali to ne mora biti razlog za odustajanje jer postojeće rješenje ne mora uvijek biti najbolje rješenje. Postoji mnogo mobilnih aplikacija s istim zadatkom, ali svaka ta aplikacija zadatak obavlja na drugačiji način.

Kako je ranije navedeno, cilj ovog završnog rada je izraditi mobilnu aplikaciju koja će nakon skeniranja QR koda pravovremenim obavijestima korisniku davati do znanja da popije propisani lijek. Na trgovini play postoji mnogo sličnih aplikacija, a najpoznatije su: MedControl, Terappia i Moj lijek. Bitno je naglasiti kako su sve one izrađene s jednim ciljem, a to je da korisnici budu posvećeni terapiji. Posvećenost terapiji znači da korisnici redovito piju lijekove i da pritom poštuju sva pravila [1]. Posvećenost je izuzetno bitna jer kako navodi farmakoterapija [2] niti jedan lijek nije djelotvoran ako se pravilno ne uzima.

Sve navedene aplikacije imaju svoje prednosti i mane te će se u narednim poglavljima detaljno usporediti s aplikacijom koja se izrađuje ovim završnim radom.

2.1. MedControl

MedControl je mobilna aplikacija dostupna za Android uređaje. Ova aplikacija pomaže da se sjetite svake tablete koju trebate uzimati bez obzira na složeni način liječenja [3]. Također, besplatna je i ne zahtjeva registraciju. To je jedna od njenih najvećih prednosti jer većina korisnika ne voli dijeliti svoje podatke kako bi mogli koristiti aplikaciju. Također, to je sličnost s aplikacijom koja se izrađuje ovim završnim radom. S druge strane, mana aplikacije MedControl je ta što se lijekovi moraju unositi ručno, a to zahtjeva puno više vremena od skeniranja QR koda.

2.2. Terappia

Terappia je mobilna aplikacija tvrtke Sandoz kojoj je cilj pomoći pacijentu da uzima lijekove redovito, na vrijeme, u ispravnoj dozi i na ispravan način [4]. Ona je namijenjena za sve vrste korisnika. Pruža izuzetno puno mogućnosti, a neke od njih su praćenje tlaka, šećera, temperature i rasta te zakazivanje podsjetnika za buduće preglede. Pružanje puno mogućnosti unutar aplikacije ima dobru i lošu stranu. Dobra strana je što korisnici mogu imati sve na jednome mjestu, a loša je što će gubiti previše vremena koristeći tu aplikaciju. Sličnost između aplikacije koja se izrađuje ovim završnim radom te aplikacije

Terappia je u mogućnosti unosa lijeka putem QR koda. S druge strane, jedna od najvećih mana aplikacije Terrapia je što za učinkovit rad zahtjeva stabilnu internetsku vezu i zbog toga ima mnogo negativnih ocjena na trgovini play.

2.3. Moj lijek

Moj lijek je vrlo jednostavna mobilna aplikacija koja omogućuje dodavanje i upravljanje različitim rasporedima lijekova [5]. Upravo zbog toga što je jednostavan čest je izbor korisnika. U usporedbi s do sada navedenim aplikacijama moj lijek je najbolje ocijenjen na trgovini play. Nedostaci aplikacije bi bili što se lijekovi unose ručno, a ne skeniranjem QR koda te oglasi koji se pojavljuju ako se ne kupi premium izdanje.

3. RAZVOJNO OKRUŽENJE

Ovo poglavlje će pobliže opisati razvojno okruženje u kojemu je aplikacija izrađena.

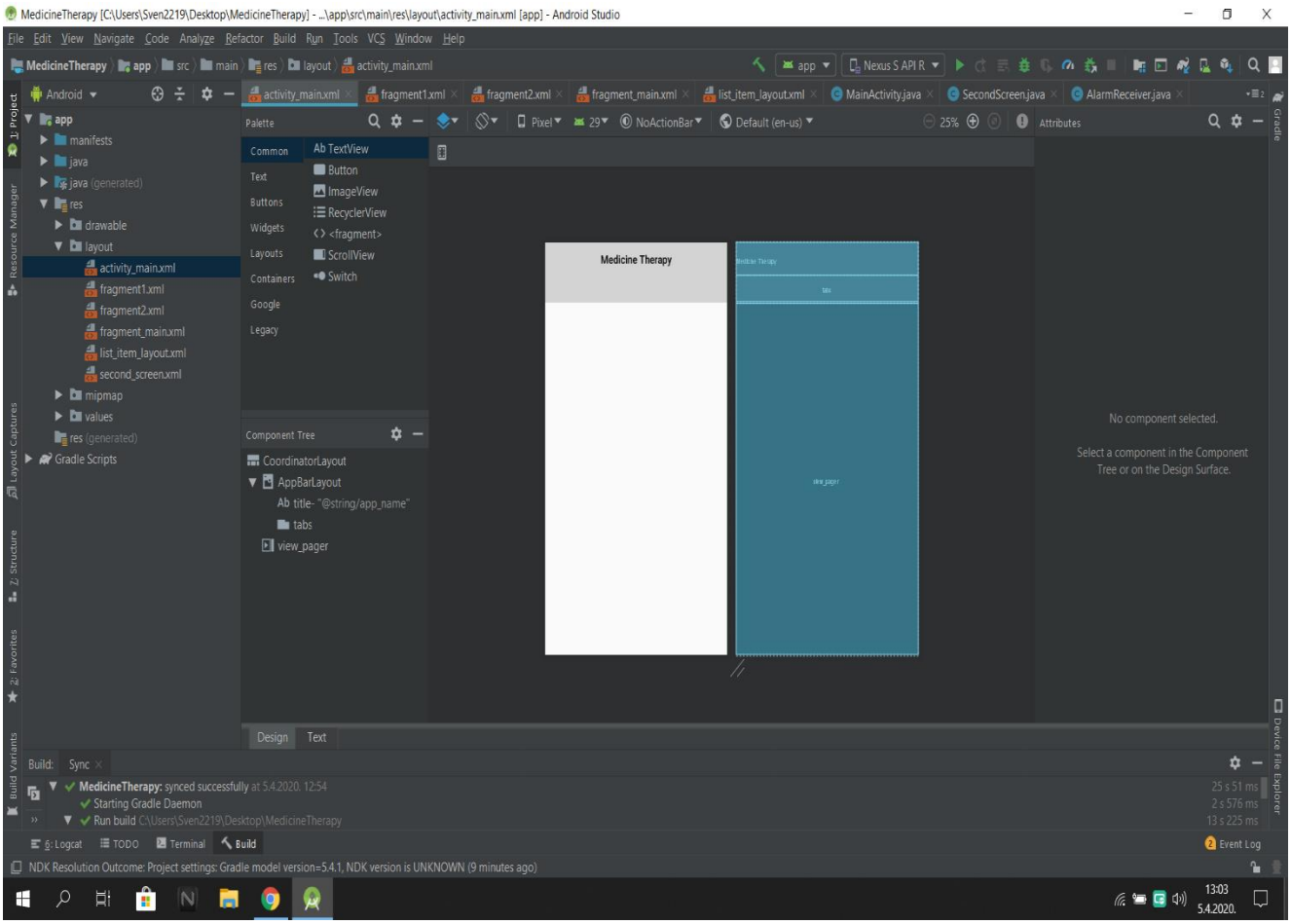
3.1. Programski jezik Java

Java je računalni programski jezik objavljen 1995. godine od strane Jamesa Goslinga i drugih inžinjera tvrtke Sun Microsystems. Objektno je orijentiran i dizajniran na način da ima što manje ovisnosti o implementaciji. Iako joj je inspiracija bio C jezik, Java koristi automatski skupljač smeća (eng. *garbage collecto*r) za upravljanje memorijom koji uvelike olakšava rad. Jedina zadaća programera je odrediti kada se objekt treba stvoriti, a Java runtime će se pobrinuti da se memorija oslobodi nakon što se objekt prestane koristiti. Zasnovana je na ideji da omogući programerima da jednom napisan kod mogu pokrenuti na bilo kojoj platformi koja podržava Javu. To znači da kod napisan i kompajliran u Microsoft Windows-u možemo koristiti i kompajlirati u Unix-u bez ikakvih potreba za modificiranjem. Do prošle godine Java je bila glavni Googlov jezik za razvoj Android aplikacija. No, Kotlin ju je zamijenio jer Google i Android tim smatraju da će razvoj Android aplikacija učiniti interesantnijim i lakšim.

3.2. Android

Android je operacijski sustav koji se temelji na Linux jezgri. Počeci razvoja vežu se uz tvrtku Android Inc, koju je Google kupio 2005. godine. Osmišljen je za mobilne uređaje kao što su pametni telefoni i tablet računala. No, Google ga je proširio na industriju automobila, satova i televizora. Trenutno je vodeći operacijski sustav za pametne telefone i tablete s gotovo 2 milijarde aktivnih korisnika mjesečno. Izvorni kod otvorenog je tipa što je omogućilo aplikacijama međusobno komuniciranje i pokretanje drugih aplikacija.

Za izradu ove aplikacije koristio se Android Studio, a njegov izgled vidimo na slici 3.1. On predstavlja službeno integrirano razvojno okruženje za razvoj Android aplikacija, temeljeno na IntelliJ IDEA [6].



Slika 3.1. *Android Studio*

4. PROGRAMSKO RJEŠENJE I PRIMJENA APLIKACIJE ZA PRAĆENJE TIJEKA TERAPIJE

4.1. Android Manifest

Obvezna stavka prilikom izrade svake Android mobilne aplikacije je postojanje AndroidManifest.xml datoteke s točno tim nazivom. Pisana je XML opisnim jezikom te predstavlja bitne informacije o aplikaciji koje koriste alati za izgradnju Android-a. Također, potrebna je radi deklaracije komponenti, samog naziva aplikacije i dozvola (eng. *permissions*) koje aplikacija mora imati kako bi mogla pristupiti zaštićenim dijelovima sustava. Prilikom izrade ove aplikacije osim deklaracije komponenti bilo je potrebno dodati brojne dozvole.

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"></uses-permission>
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"></uses-permission>
```

Slika 4.1. *Dozvole za čitanje i pisanje vanjske pohrane*

```
<uses-permission android:name="android.permission.ACCESS_NOTIFICATION_POLICY"></uses-permission>
<uses-permission android:name="android.permission.VIBRATE"></uses-permission>
<uses-permission android:name="android.permission.SET_ALARM"></uses-permission>
```

Slika 4.2. *Dozvole za notifikacije, vibracije i alarme*

```
<application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="Medicine Therapy"
```

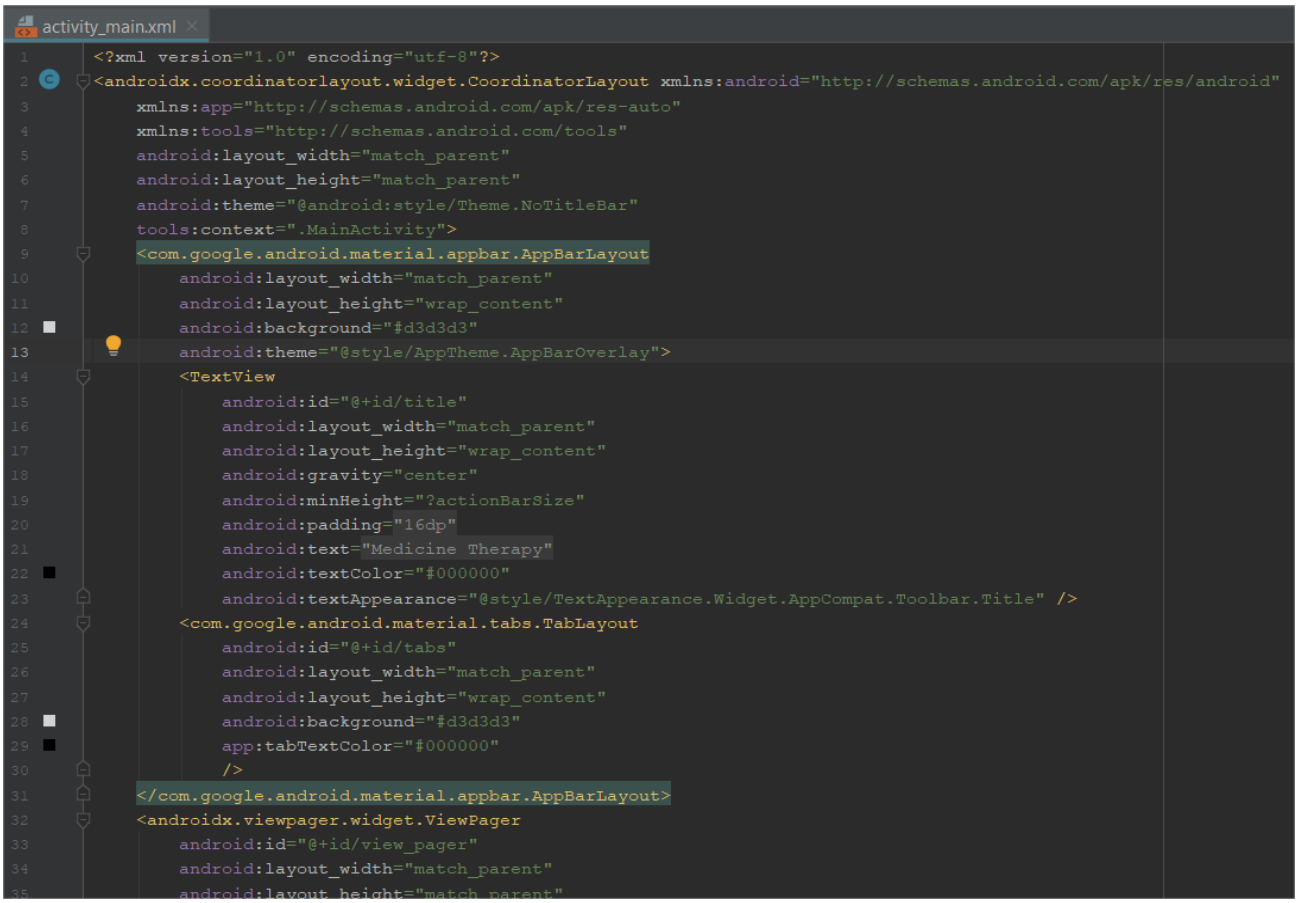
Slika 4.3. *Deklaracija imena i ikone aplikacije*

```
<activity
    android:name=".MainActivity"
    android:theme="@style/AppTheme.NoActionBar">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>

<activity android:name=".SecondScreen"></activity>
<receiver android:name=".AlarmReceiver"/>
```

Slika 4.4. *Deklaracija komponenti*

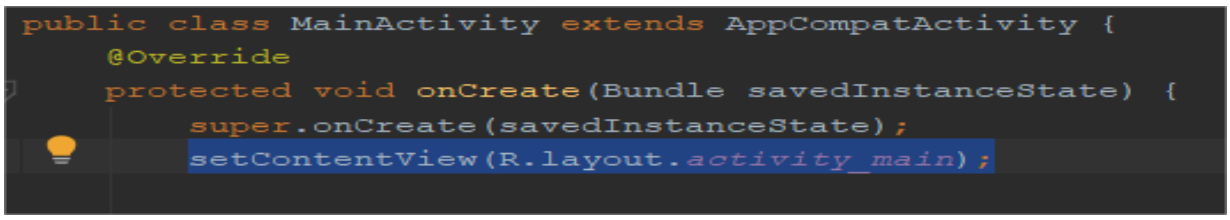
4.2. Aktivnosti i Fragmenti

Aktivnost predstavlja ključnu komponentu svake Android aplikacije. Većina aplikacija ima više zaslona što zapravo znači da sadrže više aktivnosti [7]. Općenito svaka aplikacija ima jedan glavni zaslon koji se najčešće naziva main actvitiy i on se prvi prikazuje korisniku kada pokrene aplikaciju. Izgled svakog zaslona opisan je XML opisnim jezikom, a to vidimo na slici 4.5.



Slika 4.5. *Opisivanje zaslona XML opisnim jezikom*

Gotovo svaka aktivnost u interakciji je s korisnikom pa se iz toga razloga klasa aktivnost brine za stvaranje prozora u koji se može postaviti UI koristeći metodu prikazanu na slici 4.6.



Slika 4.6. *Povezivanje Aktivnosti sa zaslonom*

Kako bi aplikacija što bolje izgledala i bila što funkcionalnija uz aktivnosti koristit će se i fragmenti. Oni postoje od samih početaka Androida. Dostupni su svim verzijama iznad 3.0. Oni su uvijek smješteni u aktivnosti, a na njihov životni ciklus izravno utječe životni ciklus aktivnosti [8]. Odnosno, kada je aktivnost pauzirana, pauzirani su svi fragmenti te aktivnosti. Svaki fragment ima jedinstveni izgled i ponašanje te ih je moguće uključiti u više aktivnosti. Za upravljanje fragmentima koristi se apstraktna klasa FragmentPagerAdapter, a klasa koja ju nasljeđuje nužna je implementirati dvije metode kako bi rad s njima bio moguć. Prva metoda sa slike 4.7 poziva se u trenutnku kada adapter zatraži fragment, a on ne postoji.

```
public Fragment getItem(int position) {
    Fragment fragment = null;
    switch (position){
        case 0:
            fragment = new Fragment1();
            break;
        case 1:
            fragment = new Fragment2();
            break;


    }
    return fragment;
}
```

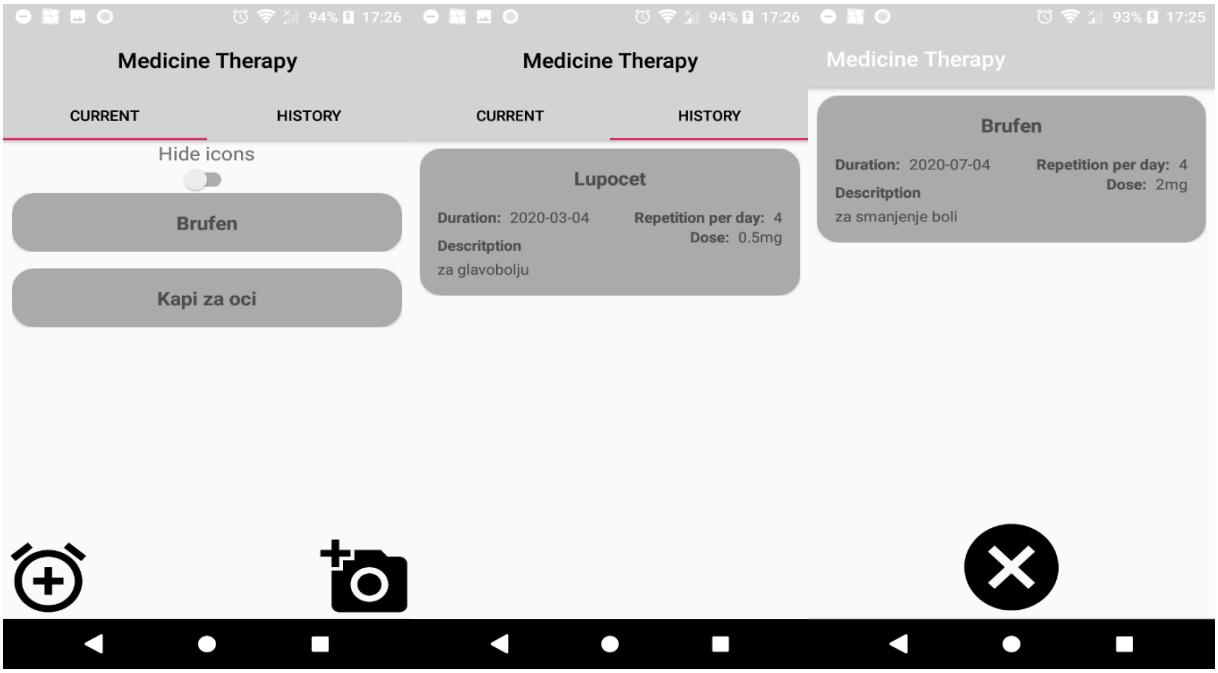
Slika 4.7. *Metoda za dohvaćanje pojedinog fragmenta*

Na slici 4.8 prikazana je druga metoda koju je potrebno implementirati i ona predstavlja broj dostupnih fragmenata.

```
public int getCount() {
    return 2;
}
```

Slika 4.8. *Metoda koja vraća broj dostupnih fragmenata*

Aplikacija sadrži dvije aktivnosti i dva fragmenta. Na slici 4.9 prikazana su tri zaslona jer fragmenti kao što je rečeno moraju biti smješteni u barem jednoj aktivnosti. U ovome slučaju oni su smješteni unutar main activitya.



Slika 4.9. *Zasloni aplikacije*

4.3. QR kod

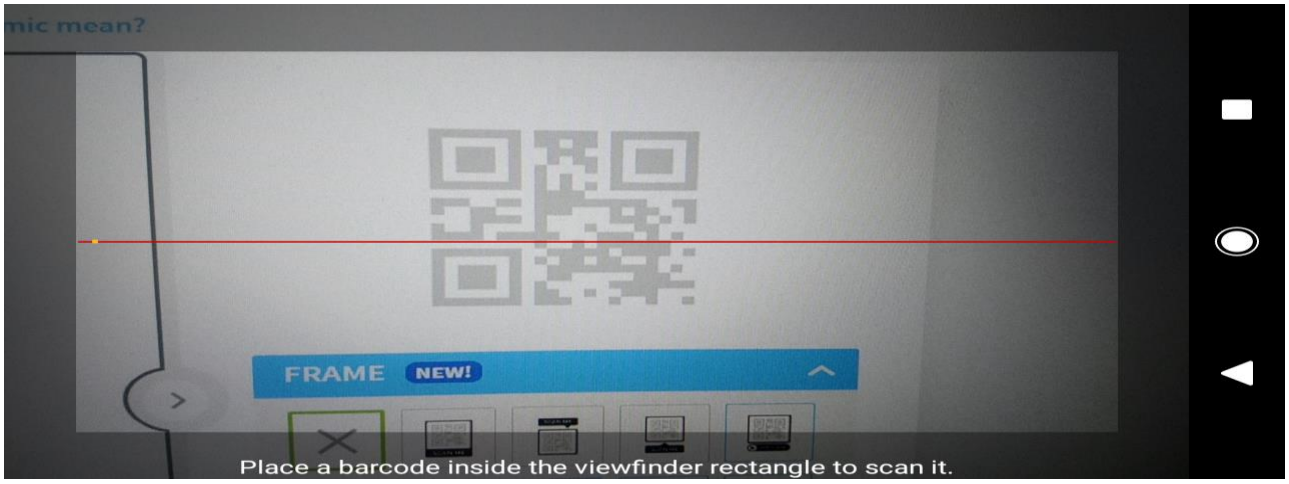
QR kod izvorno dolazi iz Japana i predstavlja skračenicu za brzi odgovor. U početku se koristio samo u automobilskoj industriji za praćenje komponenti u procesu. No, zbog svoje brzine očitavanja i mogućnosti pohrane velike količine podataka proširio se i u druge grane. Za skeniranje QR koda osim mobitela s kamerom potrebno je preuzeti aplikaciju. Na trgovini play trenutno se nalazi stotinjak QR kod čitaća.

Glavna ideja ovog završnog rada je omogućiti korisnicima brzi unos lijeka. Kako bi se izbjeglo upisivanje svih podataka ručno, korišten je QR kod. Za početak je potrebno implementirati QR kod čitać unutar aplikacije. Dekodiranje QR koda izvršavat će se ZXing bibliotekom koju je nužno dodati u dependencies unutar build.gradle. Pomoću nje je ostvaren pristup IntentIntegratoru, a on predstavlja [9] jednostavan način pozivanja na skeniranje i dobivanja rezultata, bez potrebe za učenjem izvornog koda. Kako bi se omogućilo skeniranje kreiran je objekt IntentIntegratora te je na njemu pozvana metoda initiateScan, a to je prikazano na slici 4.10. Također, na njemu se pozivaju još dvije metode. Prva metoda setBeepEnabled omogućuje zvučni signal nakon uspješnog skeniranja, a druga metoda setCameraId određuje koja kamera će se koristiti. Broj 0 predstavlja zadnju, a 1 prednju kameru.

```
final IntentIntegrator intentIntegrator = new IntentIntegrator(getActivity());
intentIntegrator.setBeepEnabled(true);
intentIntegrator.setCameraId(0);

button.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        intentIntegrator.initiateScan();
    }
});
```

Slika 4.10. *Poziv na skeniranje*

Metoda initiateScan se poziva prilikom pritiska na gumb u obliku kamere te otvara zaslon za skeniranje QR koda koji je prikazan na slici 4.11.



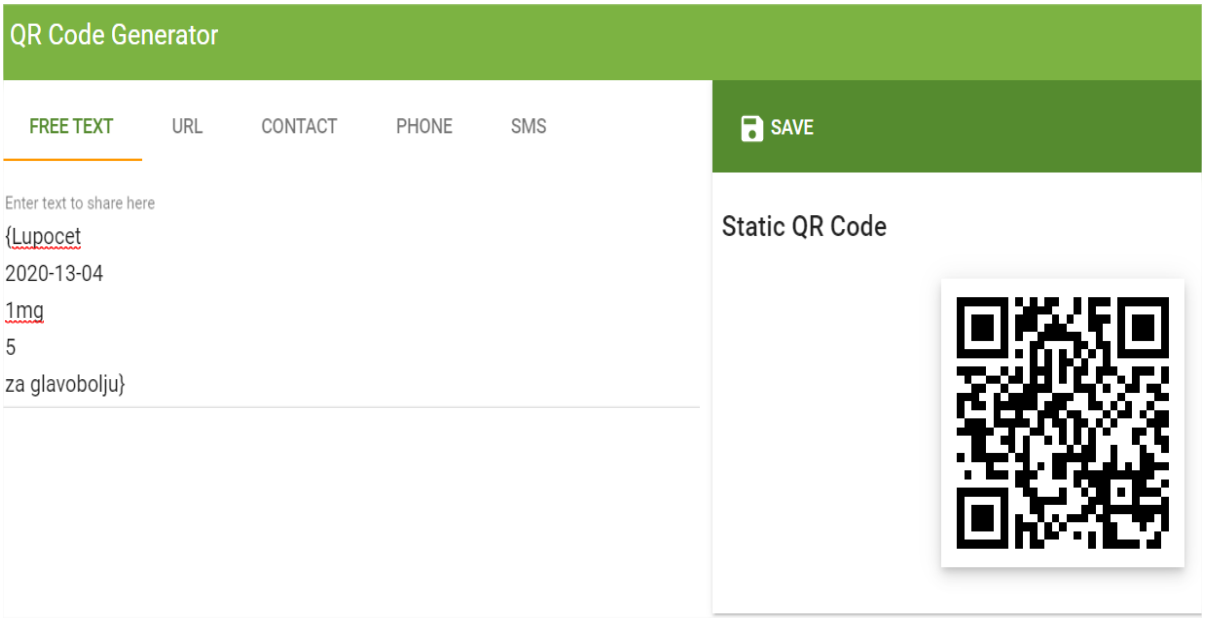
Slika 4.11. *Prikaz skeniranja*

Slika 4.12 prikazuje dohvaćanje podataka koji su upisani unutar QR koda nakon njegovog skeniranja. To je ostvareno unutar metode onActivityResult gdje se stvara objekt IntentResult koji također dolazi iz biblioteke ZXing. On je zadužen za pohranu rezultata koji je proizašao iz skeniranja. Rezultat skeniranja QR koda predstavlja niz znakova.

```
public void onActivityResult(int requestCode, int resultCode, @Nullable Intent data) {
    IntentResult result = IntentIntegrator.parseActivityResult(requestCode,resultCode,data);
    if(result!=null) {
        if (result.getContents() == null) {
            Toast.makeText(getActivity(), text: "Nothing scanned", Toast.LENGTH_LONG).show();
        } else {
            boolean inserted = therapyDatabase.tt_insertData(result.getContents());
```

Slika 4.12. *Dohvaćanje podataka iz QR koda*

QR kodovi koji se generiraju za potrebe aplikacije uvijek imaju isti format samo su podatci izmjenjeni. Njihov format je prikazan na slici 4.13.



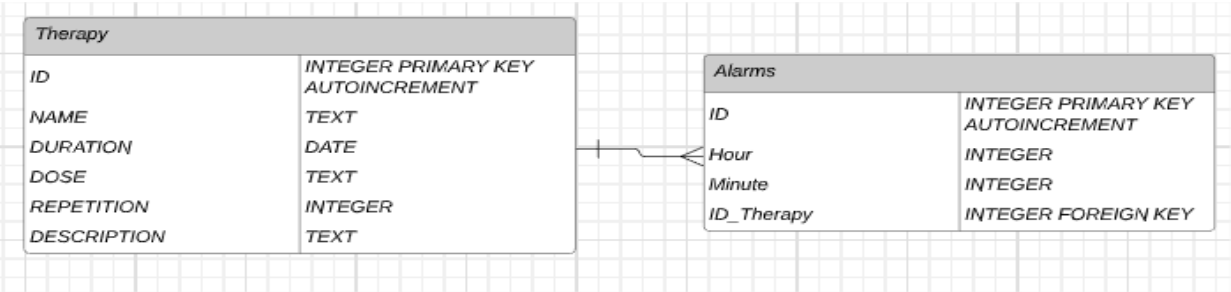
Slika 4.13. *Format QR koda*

Podatci sa slike 4.13 redom predstavljaju ime lijeka, trajanje terapije, doza lijeka, broj ponavljanja u jednom danu i opis lijeka. Kako bi se takvi podatci mogli koristiti unutar aplikacije potrebno ih je spremiti u bazu podataka.

4.4. SQLite

SQLite je biblioteka u C jeziku koja implementira samostalnu, visoko pouzdanu, potpunu značajku SQL baze podataka [10]. Ovu relacijsku bazu podataka koriste više aplikacija nego što bi smo ikada mogli nabrojati i to ju čini najkorištenijom bazom podataka na svijetu. Razlog tomu je što se kod nalazi u javnoj domeni i dostupan je svima, besplatno bez obzira u koju svrhu ga koristili. Glavne značajke zbog kojih je SQLite izabran pri izradi ove aplikacije su: trasakcije funkcioniraju po načelu ACID, odnosno one su atomske (eng. *atomic*), dosljednje (eng. *consistent*), izolirane (eng. *isolated*) i trajne (eng. *durable*) te se podatci spremaju lokalno na uređaj, a to znači da nije potreban pristup internetu prilikom njihovog dohvaćanja.

Aplikacija sadrži dvije tablice koje su međusobno povezane vezom jedan na prema više. Izgled takve veze prikazan je na slici 4.14.



Slika 4.14. *Veza između tablica te tipovi podataka*

Takva veza zapravo znači da jedna terapija ima više alarma, ali jedan alarm ima samo jednu terapiju. Obje tablice sadrže primarni ključ, a on je zadužen da na jedinstven način predstavi njihov svaki redak. Primarni ključevi se automatski popunjavaju, a to se ostvaruje ključnom riječju autoincrement. Tablica alarmi sadrži strani ključ koji predstavlja primarni ključ tablice terapija. Uz takav pristup moguće je povezati te dvije tablice i izvršavati upite nad njima.

4.4.1. Uvod u SQLite metode

Obje tablice su kreirane unutar jedne klase, a to znači da će se obje stvoriti prilikom kreiranja njenog objekta. Iz toga razloga da bi bilo jasnije koja metoda pripada kojoj tablici osmišljena je legenda koja je prikazana na slici 4.15.

```
//LEGEND

//tt=therapy table
//qott=query over two tables
//at=alarm table
```

Slika 4.15. *Legenda za bolje snalaženje unutar koda*

Svaka metoda započinje s jednom od tri mogućnosti ovisno o tome nad kojom tablicom se izvršava. Nakon toga slijedi donja crta te ime koje predstavlja što metoda zapravo radi.

```
public boolean tt_insertData(String content){
```

Slika 4.16. *Imenovanje metoda unutar klase TheraypDatabase*

Gotovo svaka metoda unutar baze podataka za zadatak ima izvršavanje SQL koda. To se može ostvariti na dva načina ovisno o tome što se očekuje kao rezultat. Prvi način je korištenje metode execSQL čiji je povratni tip void, odnosno ne vraća ništa. Drugi način je korištenje metode rawQuery koja za razliku od execSQL vraća tablicu. Metoda execSQL radi svoga ponašanja pogodna je za korištenje prilikom jednostavnijih zadataka primjerice kreiranja samih tablica.

Na slici 4.17 prikazano je kreiranje tablica koje se izvršava unutar onCreate metode jer se ona automatski poziva prilikom stvaranja objekta. Za kreiranje tablice koristi se ključna riječ CREATE TABLE te nakon nje slijedi ime tablice koja se želi kreirati. Nakon toga se otvaraju oble zagrade te se predaju imena stupaca i njihovi tipovi.

```
private static final String TABLE_NAMET = "therapytable";
private static final String T_COL_1 = "id";
private static final String T_COL_2 = "name";
private static final String T_COL_3 = "durationoftherapy";
private static final String T_COL_4 = "dose";
private static final String T_COL_5 = "repetition";
private static final String T_COL_6 = "description";
private static final String TABLE_NAMEA = "alarmtable";
private static final String A_COL_1 = "id";
private static final String A_COL_2 = "hour";
private static final String A_COL_3 = "minute";
private static final String A_COL_4 = "id_Therapy";
public TherapyDatabase(@Nullable Context context) { super(context, DATABASE_NAME, factory: null, version: 12); }
@Override
public void onCreate(SQLiteDatabase db) {
    //therapy table
    db.execSQL("CREATE TABLE " + TABLE_NAMET+"("+T_COL_1+" integer primary key autoincrement, "
            +T_COL_2+" text,"+T_COL_3+" text,"+T_COL_4+" text,"+T_COL_5+" integer,"+T_COL_6+" text );");

    //alarm table
    db.execSQL("CREATE TABLE " + TABLE_NAMEA+"("+A_COL_1+" integer primary key autoincrement, "
            +A_COL_2+" integer,"+A_COL_3+" integer,"+A_COL_4+" integer,"+" foreign key("+A_COL_4+") references "+TABLE_NAMEA+"("+T_COL_1+"));");
}
```

Slika 4.17. *Kreiranje tablica*

4.5. Spremanje podataka iz QR koda

Nakon kreiranja tablica i skeniranja QR koda popunjava se tablica terapija s podatcima koji su spremljeni unutar njega. Za popunjavanje tablice na objektu therapyDatabase poziva se metoda sa slike 4.18 čiji je zadatak pohraniti predani podatak unutar tablice terapija.

```
public boolean tt_insertData(String content){
    ContentValues contentValues = new ContentValues();
    int counter = 0;
    String[] columns = {T_COL_2,T_COL_3,T_COL_4,T_COL_5,T_COL_6};
    String temp="";
    for(char ch : content.toCharArray()){
        if(ch=='\n' || ch=='}'){
            contentValues.put(columns[counter],temp);
            temp="";
            counter++;
        }
        else if(ch!='{'){
            temp+=ch;
        }
    }
    SQLiteDatabase db = this.getWritableDatabase();
    long result = db.insert(TABLE_NAMET,null,contentValues);
    if(result == -1){return false;}
    else{return true;}
}
```

Slika 4.18. *Metoda za unos podataka unutar tablice terapija*

Na slici 4.18 prikazano je spremanje podatka unutar tablice terapija koje se izvršava na način da se znakovi spremaju u privremenu varijablu sve dok se ne pojavi znak za novi red '\n'. Kada se on pojavi, unutar ContentValuesa sprema se ime stupca i vrijednost privremene varijable. Nakon svakog spremanja privremena varijabla se postavlja na prazan tekst. Zadatak ContentValuesa je skladištenje skupa vrijednosti koji će se predati metodi insert. Metoda insert služi za popunjavanje tablice te je njen povratni tip boolean. On označava je li popunjavanje uspješno izvršeno. Prvi i zadnji znak unutar QR koda ne prikazuju se korisniku, oni samo označavaju njegov početak i kraj.

4.6. Prikaz lijekova

Za prikaz lijekova unutar aplikacije koristi se RecyclerView. On predstavlja komponentu koja se koristi prilikom prikazivanja velike količine podataka koje je moguće pomicati gore ili dolje [11]. Kako bi to bilo moguće on reciklira svoje elemente, a to znači da će biti stvoreno onoliko elemenata koliko je potrebno da se zaslon popuni. Unutar aplikacije koristi se na svim zaslonima jer svi zasloni moraju imati mogućnost prikazivanja velike količine lijekova. Za njegov funkcionalan rad potrebno je napraviti nekoliko stvari. Kako bi se podatci prikazali korisniku, prvo se moraju na neki način prezentirati unutar RecyclerViewa. To se postiže tako da opišemo izgled jedne komponente u listi. Opisivanje komponente odvija se u XML datoteci te što je ona jednostavnija RecyclerView će biti brži. Izgled jedne komponente u listi prikazan je slikom 4.19.

Slika 4.19. *Izgled komponente unutar liste*

Također, kako bi se smanjio prostor koji svaka komponenta zauzima korišten je ConstraintLayout te je svaki element unutar liste učinjen rastezljivim. Nakon stvaranja oni su skupljeni, odnosno vidimo samo naziv lijeka, a pritskom na ime lijeka dobivamo detaljne informacije o njemu. Izgled komponenti prije i nakon pritiska prikazan je slikom 4.20.

Slika 4.20. *Izgled komponenti prije i nakon pritiska*

Bilo koji element unutar liste može biti skupljen ili proširen. To je ostvareno na način da se unutar klase ListItem sa slike 4.21 koja sadrži detalje o svakom elementu doda atribut expanded i po defaultu postavi na laž.

```
public class ListItem implements Serializable {
    int id;
    String name;
    String dose;
    String description;
    int repetition;
    String durationOfTherapy;
    boolean expanded;
    public ListItem(int id, String name, String dose,String description,int repetition,String durationOfTherapy) {
        this.id = id;
        this.name = name;
        this.dose = dose;
        this.description = description;
        this.repetition = repetition;
        this.durationOfTherapy = durationOfTherapy;
        this.expanded = false;
    }
    public int getId() { return id; }
    public String getName() { return name; }
    public String getDose() { return dose; }
    public String getDescription(){return description;}
    public int getRepetition(){return repetition;}
    public String getDurationOfTherapy(){return durationOfTherapy;}
    public boolean isExpanded() { return expanded; }
    public void setExpanded(boolean expanded) { this.expanded = expanded; }
}
```

Slika 4.21. *Klasa ListItem*

Na slici 4.22 prikazano je osluškivanje na klik te kada se ono dogodi promijeni se stanje atributa expanded na pritisnutom elementu unutar liste, a upravo to omogućuje rastezanje i skupljanje elemenata.

```
textViewName.setOnClickListener((view) → {
        ListItem listItem = listItemArrayList.get(getAdapterPosition());
        listItem.setExpanded(!listItem.isExpanded());
        notifyItemChanged(getAdapterPosition());
});
```

Slika 4.22. *Prikaz osluškivanja na klik*

Osim ListItem-a za uspješan rad RecyclerView-a potrebni su Adapter i ViewHolder. Adapter je klasa koja služi za njegovo kreiranje i držanje podataka koji će biti prikazani, a ViewHolder se koristi tijekom recikliranja elemenata. Prilikom kreiranja objekta RecyclerView-a kroz konstruktor se šalje View koji predstavlja sastavljeni XML [11].

```
recyclerView = (RecyclerView) rootView.findViewById(R.id.recyclerView);
recyclerView.setLayoutManager(new LinearLayoutManager(getActivity()));
therapyDatabase = new TherapyDatabase(getActivity());
Date date = getInstance().getTime();
SimpleDateFormat dateFormat = new SimpleDateFormat( pattern: "yyyy-MM-dd");
String formatedDate = dateFormat.format(date);
arrayList = therapyDatabase.tt_showData(formatedDate, flag: false);
if(arrayList.size()>0){
    myAdapter = new MyAdapter(arrayList,getContext());
    recyclerView.setAdapter(myAdapter);
}
```

Slika 4.23. *Postavljanje podataka unutar RecyclerView-a*

Slika 4.23 prikazuje postavljanje RecylerView-a unutar prvog fragmenta. Potrebno je popuniti adapter podatcima te ga povezati s RecyclerView-om pomoću setAdapter metode. U prvom fragmentu RecyclerView prikazuje lijekove kojima je datum veći od trenutnog, a kako bi takav prikaz bio moguć kreira se metoda koja će izvršiti upit nad tablicom terapija. Ona za izvršavanje SQL koda koristi metodu rawQuery koja je zadužena za izvršavanje složenijih upita te kao parametre prima pitanje (eng. *query*) i zahtjev (eng. *requirement*), a njen izgled vidimo na slici 4.24.

```
public ArrayList<ListItem> tt_showData(String time,boolean flag){
    ArrayList<ListItem> arrayList = new ArrayList<>();
    SQLiteDatabase db = this.getReadableDatabase();
    Cursor cursor = db.rawQuery( sql: "Select * from "+TABLE_NAMET, selectionArgs: null);
    String tempTime="";
    if(cursor!=null){
        while (cursor.moveToNext()){
            tempTime=cursor.getString( i: 2);
            if(checkRequirement(time,tempTime)==flag){
                ListItem listItem = new ListItem(cursor.getInt( i: 0),cursor.getString( i: 1),
                        cursor.getString( i: 3),cursor.getString( i: 5),
                        cursor.getInt( i: 4),cursor.getString( i: 2));
                arrayList.add(listItem);
            }
        }
    }
    cursor.close();
    return arrayList;
}
```

Slika 4.24. *Metoda za prikaz lijekova*

Metoda sa slike 4.24 korištena je u oba fragmenta za prikazivanje lijekova u ovisnosti o datumu. Unutar nje zahtjev se ne može predati kao argument metodi rawQuery, a razlog tomu je format datuma. Unutar aplikacije koristi se format u obliku godina-mjesec-dan te se takav format međusobno

ne može uspoređivati na jednostavan način. Nakon poziva metode rawQuery unutar kursora spremljeni su svi podatci tablice terapija, a to je ostvareno ključnom riječju Select *. Podatke unutar kursora moguće je obići metodom moveToNext koja se koristi za pomicanje kursora u novi redak. Kada se kursor pomakne u novi redak dohvaća se datum lijeka pomoću get metode. Zatim se dohvaćeni datum prosljeđuje metodi sa slike 4.25 koja će ovisno o njegovoj valjanosti vratiti istinu ili laž.

```
public boolean checkRequirement(String formatedData,String tempTime){
    int Tyear = Integer.parseInt(tempTime.substring(0,4));
    int Tmonth = Integer.parseInt(tempTime.substring(5,7));
    int Tday = Integer.parseInt(tempTime.substring(8,10));
    int year = Integer.parseInt(formatedData.substring(0,4));
    int month = Integer.parseInt(formatedData.substring(5,7));
    int day = Integer.parseInt(formatedData.substring(8,10));
    if((year>Tyear) || (year==Tyear && month>Tmonth) || (year==Tyear && month==Tmonth && day>=Tday)){
        return true;
    }
    return false;
}
```

Slika 4.25. *Metoda za provjeru valjanosti datuma*

Metoda sa slike 4.25 predstavlja dio koda koji je zajednički u više metoda. Ona kao parametre prima trenutni datum i datum pojedinog lijeka. Takvi podatci se razdvajaju na godinu mjesec i dan, a to se ostvaruje metodom substring. Ona će ovisno o argumentima koji joj se predaju izvući znakove iz niza. Datum je moguće razdvojiti tom metodom jer je njegov format uvijek jednak, a to znači da se zna od kojeg do kojeg mjesta se nalazi godina, mjesec i dan. Povratni tip metode sa slike 4.25 je boolean te se ovisno o njemu i zastavici koja se predaje metodi sa slike 4.24 prikazuju podatci na jednom od fragmenata. Odnosno, ako je datum lijeka valjan rezultat metode sa slike 4.25 biti će laž te će poziv metode za prikaz lijekova unutar prvog fragmenta izgledati kao na slici 4.26.

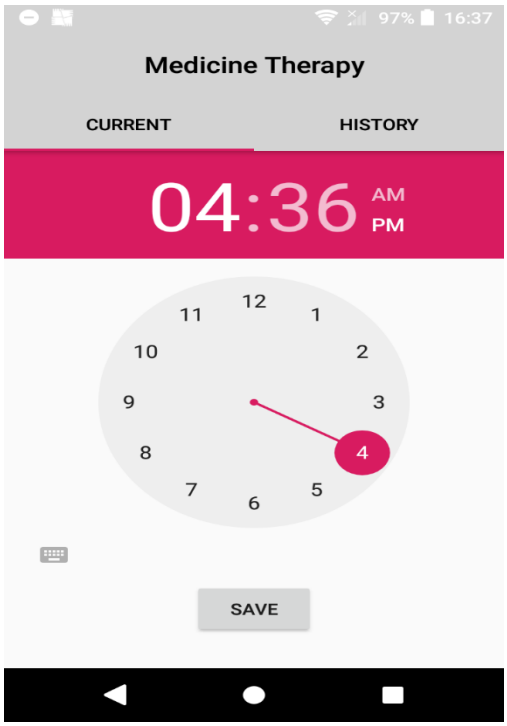
```
arrayList = therapyDatabase.tt_showData(formatedDate, flag: false);
```

Slika 4.26. *Poziv metode za dohvaćanje lijekova*

4.7. Alarmi i notifikacije

Aplikacija za obavještavanje korisnika koristi notifikacije, a kako bi ih oni dobili u točno određeno vrijeme postavljaju se alarmi. Notifikacije predstavljaju poruke koje Android prikazuje izvan korisničkog sučelja aplikacije kako bi korisniku pružio podsjetnike ili druge pravovremene

infromacije [12]. Korisnik prije skeniranja QR koda ima mogućnost postavljanja prvog alarma. Pritiskom na gumb u obliku alarma korisniku se otvara sat na kojemu može podesiti vrijeme prvog alarma. Izgled sata vidimo na slici 4.27.



Slika 4.27. *Podešavanje prvog alarma*

Nakon postavljanja vremena prvog alarma od korisnika se očekuje pritisak tipke SAVE te zatim skeniranje QR koda. Pritiskom tipke SAVE spremaju se sati i minute prvog alarma unutar dvije globalne varijable, a to je prikazano na slici 4.28.

```
saveButton.setOnClickListener((view) → {
        hourFromTimePicker = picker.getCurrentHour();
        minuteFromTimePicker = picker.getCurrentMinute();
```

Slika 4.28. *Spremanje vremena prvog alarma*

4.7.1. Postavljanje alarma

Unutar metode onActivityResult postavlja se alarm koji započinje u vrijeme kada je korisnik odredio. Ako korisnik nije postavio vrijeme prvog alarma ono se automatski postavljava u 0:0. Ključni podatak za postavljanje alarma je njegovo ponavljanje unutar jednoga dana. Taj podatak proizlazi iz QR koda

te je prilikom skeniranja spremljen u tablicu terapija. Za njegovo dohvaćanje potrebno je napraviti metodu koja će izvršiti upit nad tablicom terapija.

```
public int tt_getLastRepetition() { return getLast(rep, index: 4,TABLE_NAMET); }
public int getLast(int value,int index,String tableName){
    SQLiteDatabase db = this.getReadableDatabase();
    Cursor cursor = db.rawQuery( sql: "SELECT * FROM "+tableName+" ORDER BY id DESC LIMIT 1;", selectionArgs: null);
    if(cursor!=null){
        cursor.moveToFirst();
        value=cursor.getInt(index);
    }
    cursor.close();
    return value;
}
```

Slika 4.29. *Dohvaćanje broja ponavlja unutar jednoga dana*

Metoda getLastRepetition, prikazana na slici 4.29, unutar sebe poziva metodu getLast. Metoda getLast je stvorena radi sprječavanja dupliciranja koda, a to znači da je zajednički kod više metoda izdvojen u njoj. Ona unutar kursora pohranjuje samo jedan redak. To je ostvareno ključnom riječi ORDER BY id DESC LIMIT 1 koja je zadužena da poreda tablicu silazno po id te ograniči broj redova na jedan. Pošto je korištena u više metoda kao parametre prima globalnu varijablu unutar koje će spremiti vrijednost, broj stupca čija se vrijednost želi spremiti te ime tablice. Unutar metode getLastRepetition dohvaća se 4. stupac, a on predstavlja broj ponavljanja unutar jednoga dana. Pomoću njega se računa koliko vremena treba proći da se alarm ponovo oglasi.

```
int repetition = therapyDatabase.tt_getLastRepetition();
double hour = 24.0/repetition;
double minute = hour-Math.floor(hour);
int repetitionHour=(int)hour;
int repetitionMinute = (int) (60*minute);
```

Slika 4.30. *Računanje vremena do ponovog oglašavanja alarma*

Varijable repeatHour i repatMinute sa slike 4.30 pohranjuju sate i minute koji trebaju proći do ponovog oglašavanja alarma. Postavlja se jedan alarm s ponavljanjem u ovisnosti o njima.

Unutar aplikacije za prezentiranje vremena koristi se apstraktna klasa Kalendar (eng. *Calendar*). Metoda pomoću koje možemo inicijalizirati kalendar s trenutnim vremenom i datumom je getInstance. Nakon inicijalizacije kalendara vrijeme se može mjenjati metodama za postavljanje. Pomoću tih metoda, unutar kalendara spremit će se vrijeme prvog alarma, a to je prikazano na slici 4.31.

```
int userHour=hourFromTimePicker;
int userMinute=minuteFromTimePicker;
Calendar cal_alarm = getInstance();
cal_alarm.set(Calendar.HOUR_OF_DAY,userHour);
cal_alarm.set(Calendar.MINUTE,userMinute);
cal_alarm.set(Calendar.SECOND,0);
cal_alarm.set(Calendar.MILLISECOND,0);
long TimeUntilTrigger=cal_alarm.getTimeInMillis();
int repeatTime=repetitionHour*60+repetitionMinute;
```

Slika 4.31. *Spremanje vremena prvog alarma*

Pošto je korisniku omogućen odabir provg alarma potrebno je provjeriti je li vrijeme koje je odabrao već prošlo. Ako je vrijeme prošlo provjerava se je li moguće u tom danu izvesti alarm s obzirom na broj ponavlja u jednome danu, a ako nije alarm se pomiće za jedan dan. Provjera sa slike 4.32 izvršava se na način da se trenutno vrijeme uspoređuje s vremenom koje je korisnik odabrao. Ako je vrijeme zadano od strane korisnika manje od trenutnog vremena onda se na njegovo vrijeme dodaje vrijeme ponavljanja. Usporedba završava kada je vrijeme prvog alarma veće od trenutnog vremena.

```
long TimeUntilTrigger=cal_alarm.getTimeInMillis();
int repeatTime=repetitionHour*60+repetitionMinute;
int count=0;
while(System.currentTimeMillis()>TimeUntilTrigger){
    count++;
    TimeUntilTrigger+=repeatTime*60*1000;
    if(count==repetition){
        TimeUntilTrigger=cal_alarm.getTimeInMillis()+86400000;
    }
}
```

Slika 4.32. *Provjera oglašavanja prvog alarma*

Za postavljanje ponavljajućeg alarma kao što je prikazano na slici 4.33 zadužena je klasa AlarmManager čiji je zadatak pokretanje aplikacije u nekom trenutku u budućnosti. Na njenom objektu poziva se metoda setRepating koja kao argumente osim vremena kada se treba oglasiti alarm i koliko često se treba ponavljati prima PendingIntent, odnosno referencu na token. Ona zapravo govori koji kod će se izvršiti kada se alarm oglasi. Prilikom inicijalizacije PendingIntenta drugi argument mora biti unikatan jer inače će doći do pisanja po već postojećim alarmima.

```
Intent alarmIntent = new Intent(getActivity(), AlarmReceiver.class);
alarmIntent.putExtra( name: "alarmKey",String.valueOf(idTherapy));
pendingIntent = PendingIntent.getBroadcast(getActivity(),idTherapy,alarmIntent, flags: 0);
manager.setRepeating(AlarmManager.RTC_WAKEUP,TimeUntilTrigger, intervalMillis: repeatTime*60*1000,pendingIntent);
```

Slika 4.33. *Postavljanje ponavljajućeg alarma*

Pošto je korisniku omogućeno postavljanje prvog alarma postoji velika mogućnost da postavi alarme više lijekova u isto vrijeme. U tome slučaju svaki alarm bi kreirao zasebnu notifikaciju i pozvao melodiju, a to bi prouzrokovalo oglašavanje melodije jedne preko druge i slanje više notifikacija. Kako bi se to spriječilo kreira se tablica alarmi. Za bolje razumjevanje pristupa unošenja vremena unutar tablice alarmi u nastavku će biti prikazan primjer.

Korisnik odlučuje unijeti lijek pod imenom Lupocet koji ima pet ponavljanja u jednome danu. Nakon skeniranja QR koda podatci se prvo zapisuju unutar tablice terapija. Njen izgled nakon unosa podataka prikazan je slikom 4.34.

Id	name	durationoftherapy	dose	repetition	description
1	Lupocet	13.04.2020	2mg	5	za glavobolju

Slika 4.34. *Podatci spremljeni unutar tablice terapija*

Nakon uspješnog popunjavanja tablice terapija koristi se ranije spomenuti podatak, ponavljanje unutar jednoga dana (eng. *repetition*), koji se koristio prilikom postavljanja alarma. On određuje broj unosa u tablicu alarmi. Za unos vremena unutar tablice alarmi na vrijeme koje je zadao korisnik dodaje se vrijeme ponavljanja. Nakon toga se provjerava valjanost sata i minuta, odnosno sati moraju biti manji od 24, a minute manje od 60. Nakon obavljanja provjere, unutar tablice alarmi osim sata i minuta sprema se id lijeka kojeg korisnik treba u to vrijeme popiti, a taj podatak se dohvaća na identičan način

kao i ponavljanje unutar jednoga dana. Jedina razlika je što se metodi getLast koja vraća vrijednost jednog stupca iz zadnjeg dodanog reda predaje broj 0, odnosno stupac id. Provjera i unos podataka prikazani su slikom 4.35.

```
int idTherapy = therapyDatabase.tt_getLastId();
while (repetition>0){
    userHour +=repetitionHour;
    userMinute+=repetitionMinute;
    if(userMinute>=60){
        userHour+=1;
        userMinute-=60;
    }
    if(userHour>=24){
        userHour-=24;
    }
    therapyDatabase.at_insertData(userHour,userMinute,idTherapy);
    repetition--;
}
```

Slika 4.35. *Provjera i unos vremena*

Ako pretpostavimo da korisnik nije zadao vrijeme prvoga alarma, podatci koji će biti spremljeni unutar tablice alarmi prikazani su na slici 4.36.

	Id	horus	minute	id_therapy
1	1	4	48	1
2	2	9	36	1
3	3	14	24	1
4	4	19	12	1
5	5	0	0	1

Slika 4.36. *Podatci spremljeni unutar tablice alarmi*

4.7.2. Oglašavanje alarma

Nakon instalacije aplikacije i skeniranja QR koda lijeka popunila se tablica alarmi te se postavio alarm s određenim ponavljanjem. Za njegovo oglašavanje aplikacija ne mora biti pokrenuta. On djeluju izvan aplikacije te na taj način doprinosi smanjenju potrebe za resursima. Alarm se pojavljuje u točno

određeno vrijeme te prilikom pojavljivanja pokreće klasu AlarmReceiver. Ona se koristi za kreiranje notifikacije i postavljanja zvuka.

Nakon pokretanja klase AlarmReceiver trenutno vrijeme, odnosno sati i minute predaju se metodi koja će napraviti upit nad obje tablice te kao rezultat vratiti lijekove koje korisnik treba popiti u tom trenutku, a to je prikazano slikom 4.37.

```
rightNow = Calendar.getInstance();
int currentHour = rightNow.get(Calendar.HOUR_OF_DAY);
int currentMinute = rightNow.get(Calendar.MINUTE);
ArrayList<ListItem> listItems=new ArrayList<~>();
listItems = therapyDatabase.qott_getMedicineDependOnTime(currentHour,currentMinute);
```

Slika 4.37. *Dohvaćanje lijekova koje korisnik treba popiti*

Kako je ranije spomenuto postoji mogućnost oglašavanja više alarma u istom trenutku te je zbog toga povratni tip metode sa slike 4.38 koja se poziva unutar slike 4.37, popis nizova. Metoda sa slike 4.38 izvršava upit nad obje tablice jer se ovisno o vremenu vraćaju lijekovi, a to je ostvareno ključnom riječju INNER JOIN koja je zadužena da spoji tablice prema stranom ključu.

```
public ArrayList<ListItem> qott_getMedicineDependOnTime(int hour,int minute){
    String query = "SELECT * FROM " + TABLE_NAMET+" AS tableT" + " INNER JOIN "+TABLE_NAMEA+" AS tableA"
            + " ON tableT." + T_COL_1 + " = " + "tableA."+A_COL_4
            + " WHERE tableA.hour=? AND tableA.minute=?";
    SQLiteDatabase db = this.getReadableDatabase();
    String requirement[] = new String[]{String.valueOf(hour),String.valueOf(minute)};
    Cursor cursor = db.rawQuery(query,requirement);
    ArrayList<ListItem> arrayList = new ArrayList<>();
    if(cursor!=null){
        while (cursor.moveToNext()){
            ListItem listItem = new ListItem(cursor.getInt( i: 0),cursor.getString( i: 1),
                    cursor.getString( i: 3),cursor.getString( i: 5),
                    cursor.getInt( i: 4),cursor.getString( i: 2));
            arrayList.add(listItem);
        }
    }
    cursor.close();
    return arrayList;
}
```

Slika 4.38. *Metoda koja vraća lijekove koje korisnik treba popiti*

Nakon što znamo koje lijekove korisnik treba popiti postavlja se melodja alarma. Svaki alarm je odvojen poziv te ako se na neki način ne ograniči pozivanje melodije, ona će se oglasiti onoliko puta koliko je bilo alarma u tom trenutku. Takvo ponašanje će prouzrokovati ranije spomenuti problem oglašavanja melodije jedne preko druge. Za sprječavanje takvog ponašanja, prilikom pozivanja klase AlarmReceiver prosljeđuje se id lijeka, a to vidimo na slici 4.39.

```
Intent alarmIntent = new Intent(getActivity(), AlarmReceiver.class);
alarmIntent.putExtra( name: "alarmKey",String.valueOf(idTherapy));
```

Slika 4.39. *Prosljeđivanje podatka*

Na slici 4.40 prikazano je dohvaćanje podatka unutar klase AlarmReceiver kreiranjem paketa (eng. *Bundle*). Oni se obično koriste za prosljeđivanje podataka između različitih Android aktivnosti [14]. Za dohvaćanje i slanje podataka koristi se ključ, a jedini uvjet za uspješno prosljeđivanje je da se naziv ključa podudara.

```
Bundle extras = intent.getExtras();
if(extras != null) {
    String data = extras.getString( key: "alarmKey");
```

Slika 4.40. *Dohvaćanje podataka*

Slika 4.41 prikazuje usporedbu id prvog podatka listItems-a s vrijednošću ključa. Listitems sadrži sve lijekove koje korisnik treba popiti u tome trenutku, a to znači da će vrijednost ključa biti jednaka id-u listItems-a samo prilikom poziva prvog alarma. Upravo takav pristup omogućuje da se melodija oglasi samo jednom.

```
Bundle extras = intent.getExtras();
if(extras != null) {
    String data = extras.getString( key: "alarmKey");
    int key=Integer.parseInt(data);
    if(listItems.get(0).getId()==key){
        audioPlay.playAudio(context);
    }
}
```

Slika 4.41. *Ograničavanje broja poziva melodije*

Slika 4.41 također prikazaje pokretanje melodije, a to se postiže na način da se na objektu audioPlay pozove metoda playAudio sa slike 4.42 koja će ju emitirati 20 sekundi. Meldoija alarma aplikacije jednak je melodiji alarma na korisnikovom mobitelu, a to je ostvareno klasom RingtoneManager. Ona omogućuje pristup melodijama zvona i drugim vrstama zvukova.

```
public class AudioPlay {
    public static MediaPlayer mediaPlayer;
    public static Timer timer = null;
    public static void playAudio(Context c){
        Uri notification = RingtoneManager.getDefaultUri(RingtoneManager.TYPE_ALARM);
        mediaPlayer = MediaPlayer.create(c,notification);
        if(!mediaPlayer.isPlaying())
        {
            mediaPlayer.start();
            timer= new Timer();
            timer.schedule(() → { mediaPlayer.stop(); }, delay: 20000);
        }
    }
```

Slika 4.42. *Metoda za emitiranje melodiju*

4.7.3. Kreiranje i prikaz notifikacija

Za kreiranje notifikacije koristi se oblikovni obrazac stvaranja, graditelj. Njegova zadaća je razdvojiti stvaranje složenog objekta od njegove reprezentacije, odnosno omogućuje stvaranje složenih objekata jednakim postupkom korak po korak [13]. To znači da možemo implementirati samo dijelove koji su nam potrebni.

```
NotificationCompat.Builder mBuilder =
        new NotificationCompat.Builder(context.getApplicationContext(), channelId: "notify_001");
PendingIntent pendingIntent = PendingIntent.getActivity(context,requestCode, myintent, flags: 0);
NotificationCompat.BigTextStyle bigText = new NotificationCompat.BigTextStyle();
bigText.bigText(stringBuilder);
bigText.setBigContentTitle("Take a pill");
bigText.setSummaryText("Therapy application");
mBuilder.setContentIntent(pendingIntent);
mBuilder.setSmallIcon(R.drawable.ic_message);
mBuilder.setPriority(Notification.PRIORITY_MAX);
mBuilder.setStyle(bigText);
mBuilder.setAutoCancel(true);
```

Slika 4.43. *Stvaranje objekta metodom korak po korak*

Za kreiranje teksta koji će se prikazati korisniku prilikom dolaska notifikacije koristi se pomoćna klasa “BigTextStyle” sa slike 4.43. Ona je zadužena za generiranje obavijesti koje sadrže puno teksta, a u

slučaju da obavijest ne sadrži puno teksta prikaz će biti uobičajen [15]. Sadržaj koji će notifikacija prikazati predstavlja imena lijekova, a oni su spremljeni unutar stringBuilder-a.

Spremanje imena lijekova unutar stringBuilder-a prikazano je slikom 4.44. Ono se izvršava na način da se iz listItems-a koji je tipa ArrayList<ListItem> dohvaća pojedini ListItem koji je ništa drugo nego objekt. Nakon dohvaćanja ListItem-a na njemu se poziva metoda getName koja vraća ime lijeka.

```
StringBuilder stringBuilder = new StringBuilder();
int size=listItems.size();
while(size<0){
    stringBuilder.append(listItems.get(size).getName());
    size--;
}
```

Slika 4.44. *Postavljanje imena lijekova unutar stringBuilder-a*

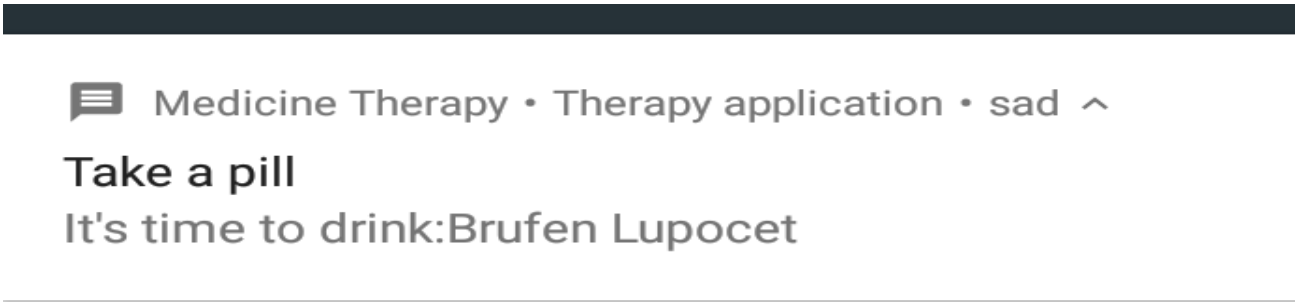
Nakon postavljanja svih željenih svojstava notifikacije provjerava se verzija Androida jer je u verziji 8.0 došlo do promjena. Ona naime zahtjeva specificiranje kanala na kojemu će notifikacije biti dostavljene.

```
mNotificationManager = (NotificationManager) context.getSystemService(Context.NOTIFICATION_SERVICE);

if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O)
{
    String channelId = "notify_001";
    NotificationChannel channel = new NotificationChannel(
            channelId,
            name: "Channel",
            NotificationManager.IMPORTANCE_HIGH);
    mNotificationManager.createNotificationChannel(channel);
    mBuilder.setChannelId(channelId);
}
mNotificationManager.notify( id: 0, mBuilder.build());
```

Slika 4.45. *Provjera Android verzije i specificiranje kanala za notifikacije*

Kod sa slike 4.45 izvršava se samo u slučaju ako je verzija Androida veća ili jednaka 8.0. U njemu se za specificiranje kanala koristi objekt notificationManager. Također, taj objekt se koristi i prilikom slanja same notifikacije te mu se kao argument predaje mBuilder.build što predstavlja izgradnju notifikacije s onim svojstvima koja su joj ranije postavljena.

Neovisno o broju alarma koji su pozvani u nekom trenutku, prikazuje se samo jedna notifikacija unutar koje se nalaze svi lijekovi koje korisnik treba popiti. Izgled notifikacije vidimo na slici 4.46.



Slika 4.46. *Izgled notifikacije*

Na pritisak notifikacije otvara se novi zaslon, a to je ostvareno pomoću PendingIntent-a. On se unutar notifikacije postavlja pomoću setContentIntenta, a to vidimo na slici 4.47.

```
PendingIntent pendingIntent = PendingIntent.getActivity(context,requestCode, myintent, flags: 0);
mBuilder.setContentIntent(pendingIntent);
```

Slika 4.47. *Postavljanje PendingIntenta unutar notifikacije*

Kao treći argument predaje se myintent unutar kojega se postavljaju lijekovi koji će biti prikazani u novome zaslonu. To se ponovo ostvaraju kreiranjem paketa, ali u ovome slučaju pošto se prosljeđuje popis nizova koristi se metoda putSerializable prikazana na slici 4.48.

```
Intent myintent = new Intent(context,SecondScreen.class);
Bundle bundle = new Bundle();
bundle.putSerializable("mylist",listItems);
myintent.putExtras(bundle);
```

Slika 4.48. *Prosljeđivanje lijekova koje korisnik treba popiti*

Prosljeđene podatke potrebno je dohvatiti i postaviti unutar RecyclerView-a kako bi se mogli prikazati korisniku. Pošto se podatci prosljeđuju metodom putSerializable za njihovo dohvaćanje potrebno je koristiti metodu getSerializable. Nakon dohvaćanja, podatci se postavljaju unutar RecyclerView-a pomoću Adaptera, a to je prikazano na slici 4.49.

```
Bundle bundleObject = getIntent().getExtras();
arrayList = (ArrayList<ListItem>)bundleObject.getSerializable( key: "mylist");
if(arrayList.size()>0){
    myAdapter = new MyAdapter(arrayList, context: this);
    recyclerView.setAdapter(myAdapter);
}
```

Slika 4.49. *Dohvaćanje i postavljanje lijekova koje korisnik treba popiti*

Dohvaćeni lijekovi se prikazuju u novome zaslonu, a njegov izgled prikazan je slikom 4.50.

4.50. *Zaslon koji se otvara na pritisak notifikacije*

Osim lijekova, prikazan će biti i gumb X te se na njegov pritisak gasi zvuk melodije i vraća na početni zaslon. Metoda koja je zadužena za navedeno prikazana je slikom 4.51.

```
private void cancelAlarm(){
    button.setOnClickListener((v) → {
            if(arrayList.size()>0){
                arrayList.clear();
                myAdapter.notifyDataSetChanged();
            }
            audioPlay.stopAudio();
            Intent intent = new Intent(getApplicationContext(),MainActivity.class);
            startActivity(intent);
    });
}
```

Slika 4.51. *Metoda za stopiranje melodije i vraćanje na početni zaslon*

4.7.4. Brisanje alarma

Brisanje alarma se provodi kada je trenutni datum manji ili jednak datumu dobivenog s QR koda, a njegova usporedba se odvija prilikom svakog pokretanja aplikacije. Osim brisanja alarma potrebno je obrisati spremljene podatke unutar tablice alarmi. Kako bi se mogao obrisati alarm i podatci iz tablice alarmi kreira se metoda sa slike 4.52. Ona unutar kursora pohranjuje podatke ovisno o pitanju (eng. *query*) koje joj se preda. Za provjeru ispravnosti podataka tablice alarmi kreira se pitanje unutar kojega se povezuju dvije tablice, a to vidimo na slici 4.53. Metoda sa slike 4.52, nakon poziva metode rawQuery unutar kursora ima pohranjene id-eve i datume koje je potrebno provjeriti. Prilikom obilaska kursora poziva se ranije spomenuta metoda sa slike 4.25 te ako je njen povratni tip istina sprema se id u listu cjelobrojnih brojeva. Njen povratni tip je istina u slučaju isteka trajanja terapije.

```
public List<Integer> retreiveInvalidId(String formatedData,String query){
    List<Integer> arrayOfId = new ArrayList<~>();
    SQLiteDatabase db = this.getReadableDatabase();
    Cursor cursor = db.rawQuery(query, selectionArgs: null);
    String tempTime="";
    if(cursor!=null){
        while (cursor.moveToNext()){
            tempTime=cursor.getString( i: 1);
            if(checkRequirement(formatedData,tempTime)) {
                arrayOfId.add(cursor.getInt( i: 0));
            }
        }
    }
    cursor.close();
    return arrayOfId;
}
```

Slika 4.52. *Metoda za dohvaćanje id-eva*

```
public List<Integer> qott_getInvalidID(String formatedDate){
    String query = "SELECT tableA.id,tableT.durationoftherapy FROM " + TABLE_NAMEA+" AS tableA,"
            +TABLE_NAMET+" AS tableT" + " INNER JOIN " + TABLE_NAMET
            + " ON tableA." + A_COL_4 + " = " + "tableT."+T_COL_1;
    return retreiveInvalidId(formatedDate,query);
}
```

Slika 4.53. *Pitanje za dohvaćanje podataka iz tablice alarmi*

Rezultat se dohvaća unutar metode sa slike 4.54, a njen zadatak je obrisati predane id-eve.

```
private void deleteDataFromAlarmTable(String formatedDate){
    List<Integer> arrayOfId = therapyDatabase.qott_getInvalidID(formatedDate);
    HashSet<Integer> hashSet = new HashSet<~>();
    hashSet.addAll(arrayOfId);
    arrayOfId.clear();
    arrayOfId.addAll(hashSet);
    int counter = 0;
    while(counter<arrayOfId.size()){
        therapyDatabase.at_deleteRow(arrayOfId.get(counter));
        counter++;
    }
}
```

Slika 4.54. *Brisanje podataka iz tablice alarmi*

Podatci se brišu metodom deleteRow kojoj se predaje id podatka koji se želi obrisati. Pošto metoda deleteRow ne izvršava nikakve upite nad podatcima unutar nje se za izvršavanje SQL koda koristi metoda execSql, a to je prikazano slikom 4.55.

```
public void at_deleteRow(int index){
    SQLiteDatabase db = this.getReadableDatabase();
    db.execSQL("DELETE FROM "+TABLE_NAMEA+" WHERE id="+index);
}
```

Slika 4.55. *Metoda zadužena za brisanje retka tablice*

Nakon brisanja podataka alarmi su još uvijek aktivni jer oni nemaju nikakvu povezanost s tablicom alarmi. Ona sprema njihovo vrijeme oglašavanja, ali ni na koji način ne utječe na njihovo pozivanje. Prilikom kreiranja alarma, PendingIntent-u kao unikatna vrijednost predavao se id lijeka. Stoga je on potreban za njegovo brisanje. Dohvaćanje id lijeka kojemu je istekao rok uzimanja izvršava se na način da se metodi sa slike 4.52 preda pitanje sa slike 4.56 koje se izvršava nad tablicom terapija.

```
public List<Integer> tt_getInvalidID(String formatedData) {
    String query = "SELECT id,durationoftherapy FROM " + TABLE_NAMET;
    return retreiveInvalidId(formatedData,query);
}
```

Slika 4.56. *Dohvaćanje id-eva lijekova kojima je istekao rok uzimanja*

Slika 4.57 prikazuje brisanje alarma, a jedine razlike između brisanja i postavljanja alarma su u pozivu metode nad objektom AlarmManager-a i zastavice koja se predaje PendingIntent-u. Prilikom kreiranja objekta PendingIntent-a kao drugi argument predaje se id lijeka kojemu je istekao rok uzimanja. No, to ne znači da je na njemu postavljen alarm. Prije samog brisanja alarma provjerava se njegovo postojenje, a za to je zadužena zastavica FLAG_NO_CREATE. Ona će kao rezultat postojanja alarma unutar boolean varijable alarmUp spremiti vrijednost istine ili laži. Ako alarm postoji na objektu AlarmManager-a poziva se metoda cancel(), a njen je zadatak obrisati alarm.

```
private void cancelAlarms(String formatedDate){
    List<Integer> arrayOfAlarmsId=therapyDatabase.tt_getID(formatedDate);
    AlarmManager manager = (AlarmManager) getActivity().getSystemService(Context.ALARM_SERVICE);
    int counter = 0;
    while(counter<arrayOfAlarmsId.size()){
        Intent cancelIntent = new Intent(getActivity(),AlarmReceiver.class);
        PendingIntent pendingCancle = PendingIntent.getBroadcast(getActivity(),arrayOfAlarmsId.get(counter),
                cancelIntent,PendingIntent.FLAG_NO_CREATE);
        boolean alarmUp = pendingCancle!=null;
        if(alarmUp){
            manager.cancel(pendingCancle);
        }
        counter++;
    }

}
```

Slika 4.57. *Brisanje postavljenih alarma*

Nakon brisanja alarma, podatci su još uvijek zapisani unutar tablice terapija. Razlog tomu je što se oni nakon isteka prikazuju na drugom fragmetnu koji predstavlja korisnikovu medicinsku povijest. Oba fragmeta prikazuju lijekove na identičan način korištenjem RecyclerView-a. Jedina razlika je u vrijednosti zastavice koja se predaje prilikom dohvaćanja lijekova. Naime, unutar drugoga fragmenta za dohvaćanje lijekova kao vrijednost zastavice predaje se istina, a to zapravo znači da se dohvaćaju samo lijekovi kojima je istekao rok uzimanja.

5. ZAKLJUČAK

Cilj ovog završnog rada je bio izraditi Android aplikaciju, koja će biti namijenjena ljudima koji moraju uzimati lijekove. Prilikom izrade aplikacije koristilo se razvojno okruženje Android Studio te programski jezik Java. Stoga je aplikacija dostupna isključivo korisnicima Android uređaja.

Nakon instalacije aplikacije korisniku je omogućeno odmah koristit aplikaciju bez ikakve potrebe za logiranjem, a za to je zadužena SQLite relacijska baza podataka. Ona sprema podatke lokalno na uređaj te aplikacija radi normalno bez prisustva interneta, a upravo je to jedna od najvećih pogodnosti aplikacije. Za brzi unos lijekova u bazu podataka korišten je čitać QR kodova. Pomoću njega se sprječava unos krivih podataka i jedina stvar koju korsinik treba napraviti da bi pratio svoju terapiju i dobivao pravovremene obavijesti je skeniranje QR koda. Također, na brzinu aplikacije utječu reciklirajući elementi kao što su fragmenti i RecyclerView koji zauzimaju onoliko memorije koliko je potrebno.

Aplikacija se testirala te se pokazala ispravnom i korisnom, odnosno nakon skeniranja QR koda oglašavali su se pravovremeni alarmi. Također, s istekom trajanja terapije lijek je otišao u korisnikovu medicinsku povijest. No, prilikom testiranja uočio se problem koji se događa kod nekolicine uređaja. Naime, neki uređaji imaju problema s primanjem obavijesti od neaktivnih aplikacija. Taj problem se rješava tako da se potvrde postavke telefona, a koraci potvrde se razlikuju ovisno o modelu uređaja. Aplikacija se daljnjim razvojem može unaprijediti uvođenjem kalendara koji će dati bolju vizualnu predodžbu trajanja terapije.

LITERATURA

[1] Uvod u antiretrovirusnu terapiju – 4.dio, http://huhiv.hr/uvod-u-antiretrovirusnu-terapiju-4-dio/ ,20.05.2020

[2] Ponašanje kroničnih bolesnika prema propisanoj terapiji, http://zdravljezasve.hr/html/zdravlje1_farmakoterapija.html, 20.05.2020

[3] MedControl app, https://tracxn.com/d/companies/medcontrol.app, 20.05.2020

[4] Tvrtka Sandoz, http://www.edborel.hr/htm/Sandoz_Terappia.htm, 20.05.2020

[5] Trgovina play, https://play.google.com/store/apps/details?id=com.devsoldiers.calendar.pills.limit&hl=hr, 20.05. 2020

[6] Android Developres, https://developer.android.com/studio/intro , 10.04.2020

[7] Android Developers, https://developer.android.com/guide/components/activities/intro-activities ,10.04.2020

[8] Kolegij: Osnove razvoja web i mobilnih aplikacija laboratorijska vježba 7, 12.04.2020

[9] Github, https://zxing.github.io/zxing/apidocs/com/google/zxing/integration/android/IntentIntegrator.html, 13.04.2020

[10] About SQLite, https://www.sqlite.org/about.html, 16.04.2020

[11] Kolegij: Osnove razvoja web i mobilnih aplikacija laboratorijska vježba 6, 19.04.2020

[12] Android Developers, https://developer.android.com/guide/topics/ui/notifiers/notifications ,28.04.2020

[13] Kolegij:Razvoj programske podrške objektno orijentiranim načelima predavanje 4, 30.04.2020

[14] Android Developers, https://developer.android.com/reference/android/os/Bundle, 30.04.2020

[15] Android Developers, https://developer.android.com/reference/android/app/Notification.BigTextStyle, 01.05.2020

SAŽETAK

U ovom se završnom radu opisuje izrada mobilne aplikacije za praćenje tijeka terapije. Aplikacija je namijenjena svim korisnicima Android uređaja koji moraju uzimati bilo koju vrstu lijekova. Temelji se na skeniranju QR koda lijeka te zapisivanju tih podataka u SQLite relacijsku bazu podataka. Nakon skeniranja QR koda, korisnik je započeo s uzimanjem terapije, što znači da se postavlja alarm koji predstavlja podsjetnik za uzimanje lijeka. Alarm se postavlja u ovisnosti o podatku s QR koda i ponavlja se tijekom jednoga dana. Osim tog podatka, unutar aplikacije korisnik može vidjeti trajanje terapije, naziv, opis i dozu lijeka. Korisnik će zaprimati obavijesti sve do isteka trajanja terapije, a s istekom lijek odlazi u njegovu medicinsku povijest. Prilikom rješavanja korišten je Android Studio te programski jezik Java. Također, korišteni su fragmenti i RecyclerView koji su odgovorni za smanjenje potrošnje resursa, a time i brzinu aplikacije.

Ključne riječi: Android aplikacija, praćenje tijeka terapije, QR kod, SQLite

ABSTRACT

Title: Android application for tracking the flow of therapy

This final paper elaborates on a mobile application for tracking the therapy. The application can be useful to all Android users required to take some medicine. The application is based on scanning a medicine QR code and recording the data in SQLite relational database. After scanning the QR code, the treatment begins and a user starts taking his/her medicine which has a set alarm as a reminder. The alarm is associated with the QR code and repeats during a day. Additionally, other relevant information, such as therapy duration, name, description and dosage, are available to the user who receives notifications during the therapy upon which everything is recorded in the medical history. The application was designed using Android Studio and Java programming language. Fragments and RecyclerView, resource management and consequently the application speed, were also used.

Key words: Android application, tracking the therapy flow, QR code, SQLlite

ŽIVOTOPIS

Sven Suk rođen je u Našicama 22.rujna 1998. godine. U Našicama završava osnovnu školu i prirodoslovno-matematičku gimnaziju s odličnim uspjehom. Nakon završteka gimnazije 2017. godine upisuje se na preddiplomski studij računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku. Cilj mu je nakon završteka preddiplomskog studija upisati diplomski studij programsko inžinjerstvo na istoimenom fakultetu. Razvoj mobilnih aplikacija mu je izuzetno zanimljiv, posebno u jezicima Java i javascript.

Sven Suk

_____

PRILOZI

Prilog 1: Završni rad .docx formata

Prilog 2: Završni rad .pdf formata

Prilog 3: Izvorni kod aplikacije