Aplikacija za Android platformu za mjerenja masnoće u hrani

Salopek, Stjepan

Undergraduate thesis / Završni rad

2017

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:595419

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-04-26**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Stručni studij

APLIKACIJA ZA ANDROID PLATFORMU ZA MJERENJA MASNOĆE U HRANI

Završni rad

Stjepan Salopek

Osijek, 2017.

Sadržaj

1. UVOD 1
2. OPERACIJSKI SUSTAV ANDROID 2
2.1. Arhitektura Androida 2
2.2. SQLite 4
3. FUNKCIONALNOST APLIKACIJE 6
3.1. Komponente aplikacije 6
3.2. Baza podataka 9
4. OPIS I RAZVOJ APLIKACIJE 13
4.1. Razvoj aplikacije 15
5. ZAKLJUČAK 21

1. UVOD

U današnje vrijeme razvoj i usluga pametnih mobilnih uređaja raste u velikoj mjeri. Svakodnevno se putem elektroničkih medija korisnicima pružaju informacije o novim, boljim i bržim rješenjima. Iako prvih šest mjeseci ove godine, veliki proizvođači mobilnih uređaja bilježe pad prodaje, razvoj je i dalje na vrhuncu. Rad i optimizacija operacijskih sustava omogućuje svakodnevni pristup novim aplikacijama. Upravo je *Android* operacijski sustav poseban jer korisnicima nudi izvorni kod (*engl. open source*). Pojam *open source* se odnosi na kod kojemu ljudi javno mogu pristupiti, te ovisno o licenci, bilo tko ga može koristiti, izmjenjivati ili dijeliti. Zadnji podaci za mjesec lipanj 2017. godine pokazuju da je u Google Play trgovini dostupno oko 3 milijuna *Android* aplikacija te taj broj svakodnevno raste. Upravo su ti brojevi najbolji pokazatelji koliko je *Android* operacijski sustav otvoren i dostupan za razvoj svim korisnicima.

Zadatak ovog završnog rada je kreirati aplikaciju za *Android* platformu koja mjeri masnoću u hrani. Aplikacija omogućava unos i izmjenu sastava pojedinog prehrambenog proizvoda te na osnovu unesene mase, aplikacija računa kolika je masa masnoće pojedine namirnice i više odabranih namirnica. U nastavku je detaljnije objašnjen rad aplikacije.

Aplikacija je napisana pomoću Android Studio programskog alata IDE (engl. *Integrated Development Environment*) s kojim u paketu dolazi i Android SDK (*engl. Software Development Kit*), razvijen pomoću Java programskog jezika.

Krajnji cilj aplikacije je omogućiti korisniku jednostavnu, brzu i točnu uslugu koja će olakšati evidenciju o unosu masnoće i ostalih nutricionističkih vrijednosti u tijelo.

U drugom poglavlju opisan je operacijski sustav *Android* i njegova arhitektura. Treće poglavlje opisuje zadane zahtjeve koje aplikacija mora ispuniti, SQL bazu podataka i komponente aplikacije. U četvrtom poglavlju je objašnjen opis i razvoj aplikacije. Te se u posljednjim poglavljima nalazi zaključak rada, korištena literatura, sažetak na hrvatskom i engleskom jeziku te životopis.

2. OPERACIJSKI SUSTAV ANDROID

Google Android Inc. je tvrtka osnovana 2003. godine te je sa svojim *Android* operacijskim sustavom trenutno vodeći sustav za mobilne uređaje. Glavni cilj osnivača kompanije Android Inc. (Andy Rubin, Rich Miner, Nick Sears, Chris White) je bio kreirati pametni mobilni uređaj koji bi bio svjestan i uzimao podatke o postavkama korisnika i o njegovoj lokaciji. 2005. godine Google je kupio Android Inc. te tim potezom ulazi u utrku sa ostalim proizvođačima operacijskih sustava za mobilne platforme (Apple, Microsoft). Od 2008. godine pa do danas, izašlo je 14 službenih verzija od kojih je danas najnovija verzija 8.0 kodnog imena *Oreo*.

Prema [1], *Android* je projekt otvorenog koda, zasnovan na Linux jezgri 2.6 i napisan u C/C++ programskom jeziku. 32-bitna *ARM* (engl. *Advanced RISC Machine*) arhitektura je baza za *Android* platformu, ali unazad četiri godine se proizvode pametni telefoni i tableti sa Intelovim procesorom podržavajući 64-bitnu platformu. Iako je zadano radno okruženje (engl. *framework*) napisano u C/C++ programskom jeziku, velika većina aplikacija je pisana Java programskim jezikom koristeći *SDK* (engl. *Software Development Kit*). Korisnik može pisati aplikacije u C/C++ programskom jeziku, ali tada se koristi *NDK* (engl. *Native Code Development Kit*).

2.1. Arhitektura Androida

Arhitektura *Androida* se može podijeliti u više slojeva. Prema [2], na samom dnu se nalazi Linux jezgra koja vodi računa o pogonskim programima (engl. *drivers*) i sustavima na nivou sklopovlja te djeluje kao među sloj između *Android* operacijskog sustava i fizičkog sklopovlja. Primjer su pogonski programi za zaslon, kameru, memoriju, zvuk, Wi-Fi te upravljanje napajanjem.

Slijedi sloj biblioteke koje su napisane u C/C++ programskom jeziku:

- Media Framework – biblioteka koja podržava snimanje i reprodukciju audio i video formata
- SQLite – biblioteka za upravljanje bazama podataka
- WebKit – engine za web preglednike
- OpenGL | ES – biblioteka za prikaz grafičkih 3D aplikacija
- Surface Manager – biblioteka za nadzor prikaza grafičkog sučelja

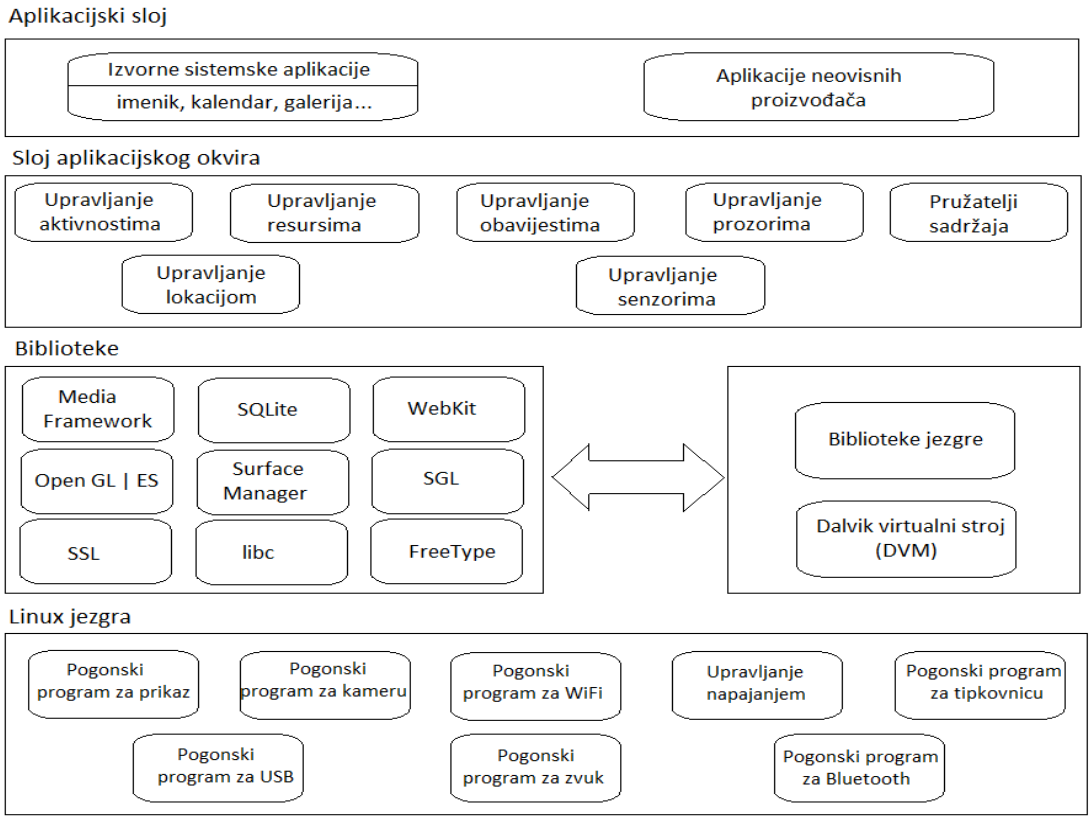
- SGL – biblioteka za implementaciju grafičkog sučelja, radi zajedno sa Surface Manager bibliotekom
- SSL – sigurnosna biblioteka za komunikaciju putem interneta
- libc – biblioteka prilagođena za rad sa sustavima baziranim na Linux operacijskom sustavu
- FreeType – biblioteka za prikaz fontova

Sljedeći sloj čini *Android* okruženje (engl. *framework*) čija je uloga pokretanje aplikacija, a sastoji se od dva dijela. Prvi dio su jezgrene biblioteke napisane u Java programskom jeziku. Drugi dio čini Dalvik virtualni stroj (engl. *Dalvik Virtual Machine*), skraćeno *DVM* čija je uloga izvršavanje više virtualnih strojeva odjednom, s ciljem iskorištenja maksimalnog potencijala Linux jezgre. To znači da se svaka aplikacija izvršava u svom virtualnom okruženju neovisno o drugim.

Sloj aplikacijskog okvira pruža mnoge usluge u obliku Java klasa koje korisniku pomažu pri pisanju aplikacija. Neke od ključnih usluga aplikacijskog okvira su :

- Upravljanje aktivnostima (engl. *activity manager*) – upravlja svim aspektima aplikacije
- Pružatelj sadržaja (engl. *content providers*) – omogućuje dijeljenje podataka jedne aplikacije sa drugim aplikacijama
- Upravljanje resursima (engl. *resource manager*) – pruža pristup ugrađenim resursima kao što su izgled korisničkog sučelja i postavke boja
- Upravljanje obavijestima (engl. *notifications manager*) – omogućuje korisniku prikaz obavijesti ili upozorenja

Na samom vrhu se nalaze aplikacije vidljive krajnjem korisniku. Tu spadaju ugrađene aplikacije poput kalendara, web preglednika, imenika, galerije. Sve aplikacije koje korisnik napiše, pa tako i ova, nalaze se u ovom sloju.



Sl. 2.1. Arhitektura operacijskog sustava Android

2.2. SQLite

SQLite je baza podataka za pohranu podataka u tekstualnom obliku koja podržava sve značajke jedne relacijske baze podataka. Usprkos svim značajkama relacijske baze podataka, SQLite se ne temelji na principu klijent-poslužitelj, nego je jednostavno ugrađen u krajnji program. Osnovne značajke relacijske baze podataka prema [3] su:

- Transakcije koriste niz pravila po načelu ACID (Atomicity, Consistency, Isolation, Durability), zbog toga su podaci sigurni u slučaju nestanka električne energije ili padu sustava
- Ne zahtjeva administraciju, konfiguraciju, instalaciju ili slično
- Podržava bazu podataka veličine reda TB (engl. *Terabyte*)
- Jednostavno korištenje API sučelja (engl. *Application Programming Interface*)
- Podržava gotovo sve platforme (Android, iOS, Linux, Mac, Solaris, Windows)

Operacijski sustav *Android* dolazi sa implementiranom SQLite bazom podatka u obliku paketa *android.database.sqlite*, te je zbog toga dovoljno samo pozvati metodu *openOrCreateDatabase* koja prima željeno ime za bazu podataka i pristupni modifikator *private, public* ili *protected.*

```
SQLiteDatabase myDatabase = openOrCreateDatabase("database name",
                MODE_PRIVATE, null);
```

SQLiteDatabase je klasa koja nudi usluge za upravljanje nad bazom podatka, odnosno pruža metode *insert( ), update( )* i *close( ).* Za kreiranje ili unos podataka koristi se metoda *execSQL*, definirana u klasi *SQLiteDatabase*. Ova metoda će također ažurirati postojeće podatke u bazi podataka.

3. FUNKCIONALNOST APLIKACIJE

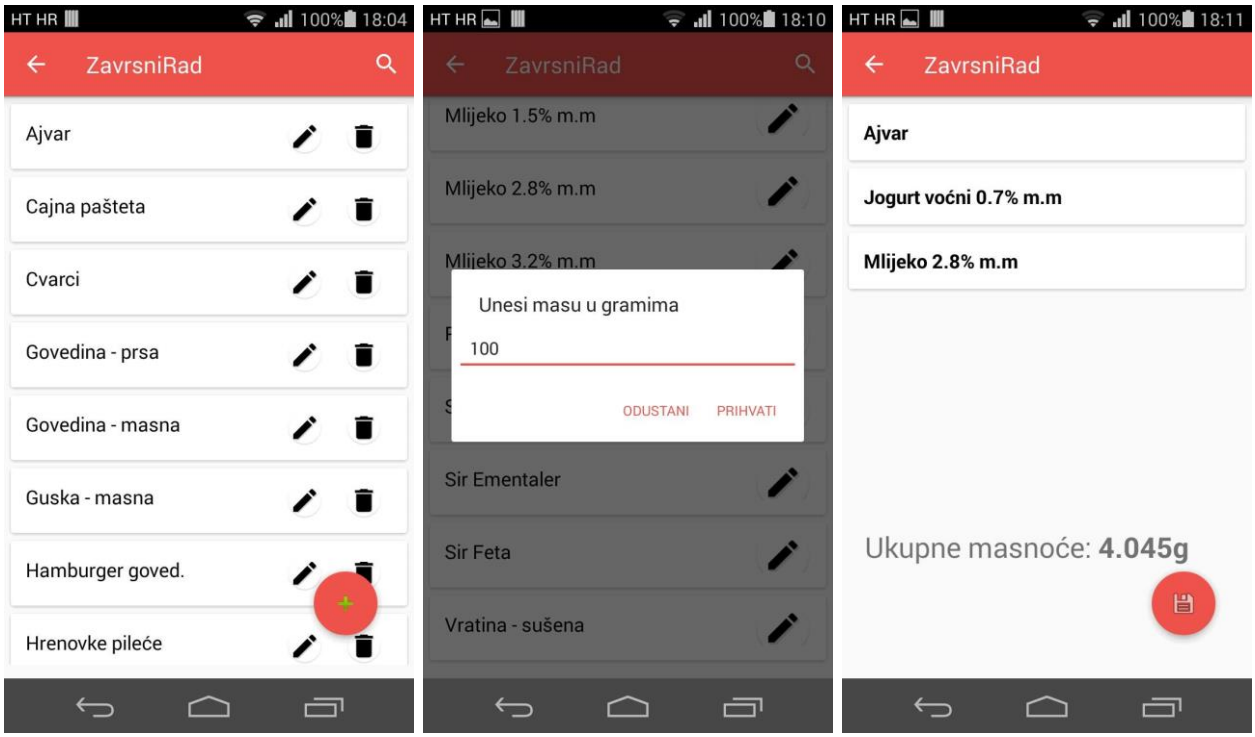
Prije izrade aplikacije postavljeni su zahtjevi koji moraju biti ispunjeni da bi aplikacija bila funkcionalna i razumljiva krajnjem korisniku. Glavne funkcionalnosti aplikacije su:

- Prikaz početnog zaslona – omogućiti prikaz početnog zaslona nakon pokretanja aplikacije gdje se korisniku pruža izbor između tri aktivnosti (menu, popis namirnica, povijest)
- Prikaz zaslona za odabir namirnica i uvid u njene vrijednosti – potrebno je omogućiti popis svih namirnica iz baze nakon što korisnik odabere stavku doručak, ručak ili večera te prikaz nutricionističkih vrijednosti nakon odabira pojedine namirnice
- Prikaz zaslona sa izračunom masnoće – nakon odabira pojedine namirnice, omogućiti korisniku unos gramaže nakon čega se na zaslonu ispisuje ukupna masnoća u gramima na osnovu unesene gramaže
- Napredno pretraživanje – pri odabiru namirnica, omogućiti korisniku napredno pretraživanje u smislu, ako korisnik u tražilicu upiše 'ml', dobiti će listu svih namirnica u bazi koje započinju slovima 'ml'
- Prikaz unesene hrane u proteklom vremenu – potrebno je omogućiti prikaz unesene masnoće u proteklom vremenu

3.1. Komponente aplikacije

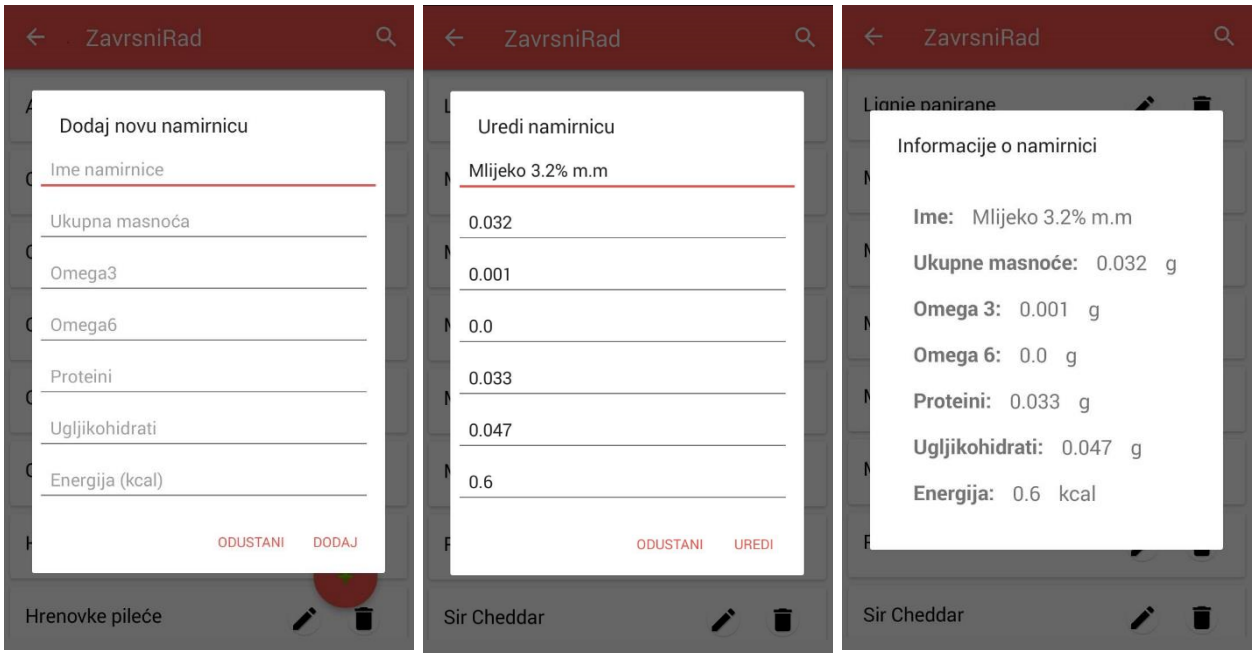
Početni zaslon se sastoji od tri gumba (engl. *button*) Menu, Popis namirnica i Povijest. Pritiskom na pojedini gumb se otvara novi zaslon sa svojim komponentama.

Unutar aktivnosti Menu nalaze se tri gumba (Doručak, Ručak i Večera) za izbor obroka. Pritiskom na jedan od ponuđenih obroka otvara se zaslon sa svim namirnicama u bazi podataka i *Floating Action Button* te korisnik ovdje ima mogućnost unos gramaže za pojedinu ili više namirnica. *Floating Action Button* otvara novi zaslon koji ispisuje one namirnice za koje je korisnik unio gramažu te se na dnu zaslona unutar elementa *TextView* ispisuje ukupna masnoća u gramima za odabrane namirnice.



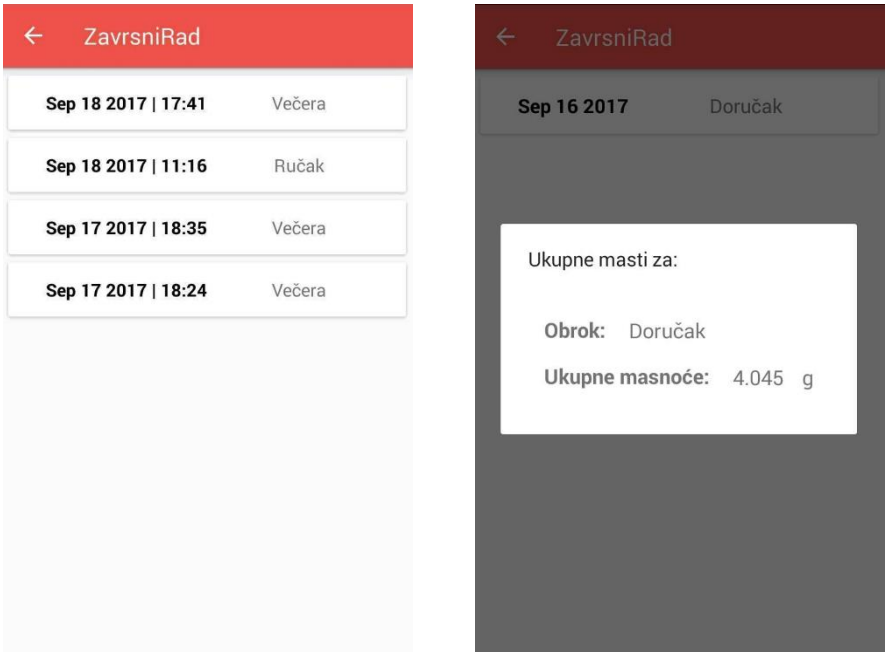
Sl. 3.1. a) Popis namirnica; **b)** Unos mase za odabranu namirnicu; **c)** Ukupne masti u gramima

Gumb 'Popis proizvoda' otvara zaslon koji prikazuje listu svih podataka koji su spremljeni u bazi podataka. Gumb za dodavanje novih namirnica je definiran kao *Floating Action Button*, te on otvara zaslon za unos nove namirnice u bazu podataka. Nakon što korisnik unese podatke, klikom na gumb 'Dodaj' se poziva funkcija *saveFood* unutar koje se uneseni podaci spremaju kao *String*. Detaljniji postupak spremanja vrijednosti u bazu podataka objašnjen je u poglavlju 3.3 Baza podataka. Korisnik ovdje ima mogućnost izmjene nutricionističkih vrijednosti za svaku namirnicu u bazi podataka. Pritiskom na ime namirnice otvara se skočni prozor koji prikazuje sve informacije o namirnici.



Sl. 3.2. a) Dodavanje nove namirnice; **b)** Uređivanje namirnice; **c)** Informacije o namirnici

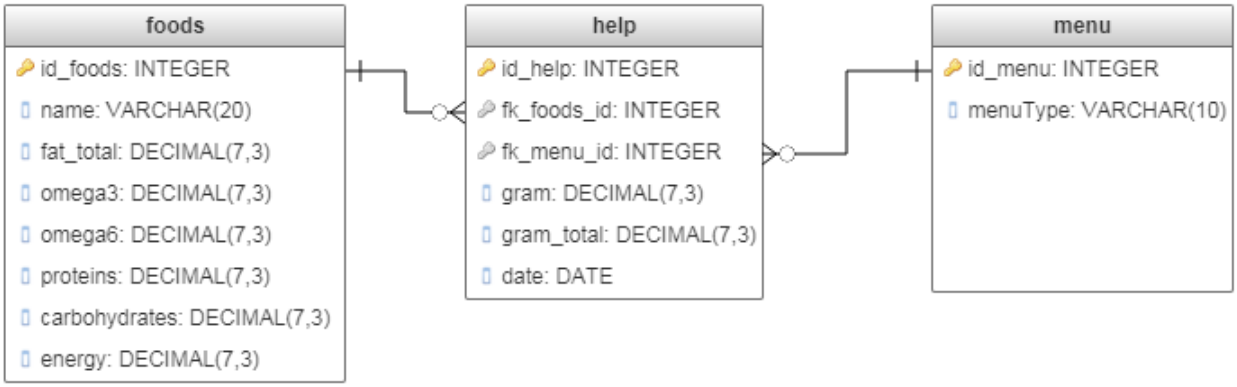
Posljednji gumb unutar početnog zaslona je Povijest te se pritiskom na njega otvara zaslon sa listom vremena. Prilikom odabira jednog vremena iz liste otvara se skočni zaslon koji ispisuje ukupne masnoće unesene u tom trenutku za odabrani obrok.



Sl. 3.3. a) Popis vremena; **b)** Prikaz ukupne masnoće za odabrano vrijeme

3.2. Baza podataka

Kao kod izrade svake baze podataka, prvo je potreba shema (engl. *schema*), drugim riječima, potrebno je definirati strukturu baze podataka. Pri tome se misli na ime tablice, povratne tipove i nazive stupaca.



Sl. 3.4. Shema baze podataka

Nadalje se definira klasa *ContractClass* koja služi kao mjesto gdje se pohranjuju imena tablica, stupaca, URI (engl. Uniform Resource Identifier), MIME tipovi podataka (tekst, slika, video, zvuk) te ostali meta podaci. Upravo se ti podaci pozivaju više puta tijekom pisanja koda, te mogućnost da se mogu izvući i pohraniti u jednu konstantu smanjuje vjerojatnost pogreške pri pisanju i omogućuje lakše održavanje za buduće promjene. U ovom slučaju se u *FoodEntry* klasu spremaju id, ime tablice te nutricionističke vrijednosti (ukupne masnoće, omega 3 i omega 6 masne kiseline, proteini, ugljikohidrati i energija).

```
public final static String _ID_FOODS = "id_foods";
public final static String TABLE_NAME = "foods";
public final static String COLUMN_FOOD_NAME = "name";
public final static String COLUMN_FAT_TOTAL = "fat_total";
public final static String COLUMN_OMEGA3 = "omega3";
public final static String COLUMN_OMEGA6 = "omega6";
public final static String COLUMN_PROTEINS = "proteins_total";
public final static String COLUMN_CARBOHYDRATES = "carbohydrates_total";
public final static String COLUMN_ENERGY = "energy";
```

Kasnije su definirane i ostale vrijednosti poput imena za druge dvije tablice, id za privatne i strane ključeve, datum, gramaža za pojedinu namirnicu te ukupna gramaža za pojedini obrok.

Nakon što je definiran izgled baze podataka, potrebno je implementirati metode za kreiranje i održavanje baze podataka i njenih tablica. Prema [4], tu ulogu preuzima klasa *SQLiteOpenHelper* koja omogućuje rukovanje nad bazom podataka samo onda kada je zatražen upit za to. Pozivanje ove klase zahtjeva od sustava potencijalno duge operacije, te se upravo zbog toga klasa izvršava samo kada je pozvana, a ne prilikom svakog pokretanja aplikacije. *SQLiteOpenHelper* klasa nudi deset javnih metoda za rad od kojih je nekoliko korišteno u ovom radu. Metoda *onCreate()* se poziva kada je baza podataka prvi put stvorena te se u nju upisuju SQL naredbe za kreiranje nove tablice. Metoda *onUpgrade()* se poziva kada je potrebno ažurirati bazu podataka, odnosno kada je potrebno stvoriti novu ili obrisati postojeću tablicu. Nadalje, metode *getWritableDatabase()* i *getReadableDatabase()* se pozivaju ovisno o tome zahtjeva li se pisanje u bazu podataka ili samo čitanje njenih vrijednosti.

```
String SQL_CREATE_FOOD_TABLE = "CREATE TABLE " + FoodEntry.TABLE_NAME + " ("
                + FoodEntry._ID_FOODS + " INTEGER PRIMARY KEY AUTOINCREMENT, "
                + FoodEntry.COLUMN_FOOD_NAME + " TEXT NOT NULL, "
                + FoodEntry.COLUMN_FAT_TOTAL + " DECIMAL(7,3) NOT NULL, "
                + FoodEntry.COLUMN_OMEGA3 + " DECIMAL(7,2), "
                + FoodEntry.COLUMN_OMEGA6 + " DECIMAL(7,2), "
                + FoodEntry.COLUMN_PROTEINS + " DECIMAL(7,2), "
                + FoodEntry.COLUMN_CARBOHYDRATES + " DECIMAL(7,3), "
                + FoodEntry.COLUMN_ENERGY + " DECIMAL(7,3)" + ")";
        db.execSQL(SQL_CREATE_FOOD_TABLE);
```

U ovom primjeru je stvorena tablica *foods* čije ime kao i imena svih stupaca poziva *Contract* klasu koja je ranije definirana. Nadalje se dodjeljuju povratni tipovi za svaki stupac, pa se tako nutricionističkim vrijednostima dodjeljuje decimalna vrijednost u obliku (7,3) i (7,2) što znači da je moguće najviše zapisati četveroznamenkasti broj sa tri odnosno dvije decimale poslije decimalne točke. Na kraju svake SQL naredbe se poziva metoda *execSQL* koja izvršava zadanu SQL izjavu.

Nakon što je stvorena tablica i njeni stupci, potrebno je unijeti podatke u nju. Ovdje se korisniku pruža mogućnost unosa svojih željenih vrijednosti, ali aplikacija dolazi sa već unesenim vrijednostima u tablici s kojima korisnik može raspolagati. Za unos vrijednosti u bazu podataka prvo se poziva metoda *getWritableDatabase*() te se nakon toga dohvaća objekt *ContentValues* koji prema [5] služi za pohranu niza vrijednosti koje se kasnije mogu obraditi. Ključnom riječi *put*, zadana vrijednost se dodaje skupu *ContentValues*. Na posljetku se poziva *insert()* metoda sa svojim argumentima te se kao prvi argument dohvaća ime tablice. Drugi argument definira što se događa u slučaju da vrijednosti nisu unesene u *ContentValues*, ali u ovom slučaju se dohvaća vrijednost *null*, te se kao treći argument dohvaća *ContentValues*. U primjeru je definiran način za unos namirnice u bazu podataka koristeći *insert()* metodu.

```
public void addFood(Food food) {
        SQLiteDatabase db = this.getWritableDatabase();
        ContentValues values = new ContentValues();
        values.put(FoodEntry.COLUMN_FOOD_NAME, food.getName());
        values.put(FoodEntry.COLUMN_FAT_TOTAL, food.getFat());
        values.put(FoodEntry.COLUMN_OMEGA3, food.getOmega3());
        values.put(FoodEntry.COLUMN_OMEGA6, food.getOmega6());
        values.put(FoodEntry.COLUMN_PROTEINS, food.getProteins());
        values.put(FoodEntry.COLUMN_CARBOHYDRATES, food.getCarbo());
        values.put(FoodEntry.COLUMN_ENERGY, food.getEnergy());
        db.insert(FoodEntry.TABLE_NAME, null, values);
    }
```

U slučaju kada korisnik unosi svoje namirnice princip je vrlo sličan, razlika je samo u tome što se vrijednosti koje korisnik unese pohranjuju u *String* konstantu. Vrijednosti se unose uz pomoć elementa *EditText* koji služi za unos i izmjenu teksta. Nadalje, korištenjem javnih metoda *getText()* i *toString()*, uneseni tekst se pohranjuje u *String* konstantu koja se dohvaća kada se pozivaju *ContentValues* vrijednosti.

Za brisanje proizvoda iz baze podataka potrebno je dodijeliti kriterije za odabir za određene retke. Prema [6], potrebno je definirati *String* argumente *selection* i *selectionArgs* nakon što se pozove metoda *getWritableDatabase()*. *Selection* argument vrača onaj redak u tablici nad kojim se obavlja radnja te isto tako nudi mogućnost kombinacije više redaka. To se izvršava tako da se pozove operator *LIKE ?*. Metoda *delete()* ima tri argumenta, a to su ime tablice, *whereClause* koji vrača odabrani redak u tablici i *whereArgs*.

```
public void deleteFood(int id) {
        SQLiteDatabase db = this.getWritableDatabase();
        db.delete(FoodEntry.TABLE_NAME, FoodEntry._ID_FOODS + "=?",
                new String[]{String.valueOf(id)});
    }
```

Za ažuriranje vrijednosti u bazi podataka koristi se *update()* metoda koja koristi sintaksu *insert()* metode sa argumentom *whereArgs* iz sintakse *delete()* metode. Navedeni argument se koristi u slučaju kada se žele ažurirati samo odabrani redci, inače se ažuriraju svi redci u tablici.

```
public void updateFood(Food food) {
        ContentValues values = new ContentValues();
        values.put(FoodEntry.COLUMN_FOOD_NAME, food.getName());
        values.put(FoodEntry.COLUMN_FAT_TOTAL, food.getFat());
        SQLiteDatabase db = this.getWritableDatabase();
        db.update(FoodEntry.TABLE_NAME, values, FoodEntry._ID_FOODS + "= ?",
                new String[]{String.valueOf(food.getId())});
    }
```

Osim spomenute tablice *foods* kreirane su još dvije tablice u bazi podataka. U tablici *menu* su pohranjeni stupci *id_menu* i *dayTime*. Uz pomoć SQL naredbe INSERT INTO unutar stupca *dayTime* su pohranjena tri retka (*breakfast*, *lunch*, *dinner*) te svaki redak ima svoj jedinstveni

primarni ključ. Upravo se taj privatni ključ povezuje sa ostalim tablicama te se dohvaća kada korisnik unutar zaslona 'Menu' odabire između doručka, ručka i večere.

Tablica *help* je kreirana za pohranu mase u gramima za svaku odabranu namirnicu, ukupnu masu za pojedini obrok te pohranu vremena. Također su stvoreni i strani ključevi *fk_foods_id* i *fk_menu_id* s kojima je tablica povezana sa ostale dvije tablice u bazi podataka.

```
String HELPER_TABLE = "CREATE TABLE " + FoodEntry.TABLE_NAME_HELPER + " ("
            + FoodEntry._ID_HELPER + " INTEGER PRIMARY KEY AUTOINCREMENT, "
            + FoodEntry.FOREIGN_FOODS_ID + " INTEGER, "
            + FoodEntry.FOREIGN_MENU_ID + " INTEGER, "
            + FoodEntry.COLUMN_GRAM + " DECIMAL(7,3), "
            + FoodEntry.COLUMN_GRAM_TOTAL + " DECIMAL(7,3), "
            + FoodEntry.COLUMN_DATE + " DATE, "
            + " FOREIGN KEY (" + FoodEntry.FOREIGN_FOODS_ID + ") REFERENCES "
            + FoodEntry.TABLE_NAME + "(" + FoodEntry._ID_FOODS + "),"
            + " FOREIGN KEY (" + FoodEntry.FOREIGN_MENU_ID + ") REFERENCES "
            + FoodEntry.TABLE_NAME_MENU + "(" + FoodEntry._ID_MENU + ")"
            + ")";
      db.execSQL(HELPER_TABLE);
```

U slučaju kada korisnik odabire između doručka, ručka ili večere unutar aktivnosti *MenuActivity*, poziva se funkcija *listMenuType*. Funkcija koristi *switch case* naredbu uz pomoć koje se poziva željeni *id_menu* ovisno o tome je li korisnik odabrao aktivnost doručak, ručak ili večera. Prikazan je primjer kako je izvedena funkcija. U slučaju da korisnik pritisne gumb Doručak poziva se slučaj 1 koji će odabrati *id_menu* iz tablice *menu* gdje je unutar stupca *dayTime* pohranjen redak *breakfast*. U situaciji kada korisnik pritisne gumb Ručak ili Večera uz pomoć naredbe *break* preskače se prethodni slučaj i poziva onaj koji odgovara odabranoj aktivnosti. Odabrani slučaj, odnosno privatni ključ se pohranjuje u tablici *help* unutar stupca *fk_menu_id*.

```
public int listMenuType(int menu) {
        String sql = "";
        switch (menu) {
            case 1:
                sql = "SELECT " + FoodEntry._ID_MENU + " FROM " + FoodEn-
try.TABLE_NAME_MENU + " WHERE " + FoodEntry.FOOD_TIME + " = 'breakfast' ;";
                break;
            case 2:
                sql = "SELECT " + FoodEntry._ID_MENU + " FROM " + FoodEn-
try.TABLE_NAME_MENU + " WHERE " + FoodEntry.FOOD_TIME + " = 'lunch' ;";
                break;
            case 3:
                sql = "SELECT " + FoodEntry._ID_MENU + " FROM " + FoodEn-
try.TABLE_NAME_MENU + " WHERE " + FoodEntry.FOOD_TIME + " = 'dinner' ;";
                break;
        }}
```

4. OPIS I RAZVOJ APLIKACIJE

Prilikom pokretanja aplikacije učitava se početni zaslon sa gumbima Menu, Popis namirnica i Povijest.

Nakon što je deklariran objekt *Button*, poziva se metoda *onClick* koja detektira svaki korisnikov dodir unutar određenog područja koji je zauzeo pogled (engl. *View*), na zaslonu uređaja. Unutar metode *onClick* se poziva operacija *Intent* koja prema [7] prima dva parametra. Prvi parametar je *Context* sučelje, a drugi parametra je klasa kojoj pristupa *Intent* (u ovom slučaju je to novi zaslon koji se otvara).

```
Button menuButton = (Button) findViewById(R.id.btnMenu);
       menuButton.setOnClickListener(new View.OnClickListener() {
           @Override
           public void onClick(View view) {
           Intent intent = new Intent(MainActivity.this, MenuActivity.class);
                startActivity(intent);
           }
       });
```

Unutar aktivnosti Menu nalaze se tri gumba (Doručak, Ručak i Večera) za izbor obroka. Pritiskom na jedan od ponuđenih obroka otvara se zaslon sa svim namirnicama u bazi podataka i *Floating Action Button* te korisnik ovdje ima mogućnost unos gramaže za pojedinu namirnicu. Unesena gramaža za svaku namirnicu se pohranjuje u tablici *help* unutar stupca *gram*.

Floating Action Button otvara novi zaslon koji ispisuje one namirnice za koje je korisnik unio gramažu te se na dnu zaslona unutar elementa *TextView* ispisuje ukupna masnoća u gramima za odabrane namirnice. U ovom trenutku korisnik ima mogućnost vratiti se natrag i unijeti novu masu za neki drugi obrok ili spremiti ukupnu masnoću u tablici *help* unutar stupca *gram_total*.

Korisnik naravno ima mogućnost potpune izmjene i uvid u sve prehrambene proizvode koji se nalaze u bazi podataka, a ta mogućnost se ostvaruje pritiskom na gumb Popis namirnica. Izmjena i brisanje namirnica je ostvareno pritiskom na odabrani gumb koji se nalazi uz svaku namirnicu u listi, a dodavanje novih namirnica se postiže uz pomoć *Floating Action Button*-a. Pritiskom na gumb za izmjenu podataka otvara se skočni zaslon gdje korisnik ima mogućnost izmijeniti postavljene ili prethodno unesene nutricionističke vrijednosti. Nadalje se poziva metoda *update* koja je definirana unutar *FoodDbHelper* klase.

Dodavanje nove namirnice je ostvareno na isti način kao i izmjena, ali se u ovom slučaju poziva metoda *insert*. Navedene metode su detaljnije objašnjene u podpoglavlju 3.3 Baza podataka.

Korisnik također ima na izbor korištenja naprednog pretraživanja. Napredno pretraživanje koristi *SearchView widget* koji omogućuje ispis svih namirnica ovisno o tome koja slova je korisnik unio, ako na primjer korisnik unese slova 'ml' ispisat će se sve namirnice koje započinju tim slovima. Za točnije pretraživanje korisnik naravno može unijeti riječ ili dio riječi kako bi si skratio izbor.

Prema [8], *SearchView* je *widget* koji korisniku pruža sučelje za unos upita za pretraživanje. Prilikom unosa slova, *SearchView* prikazuje popis prijedloga ili rezultata, ako su oni dostupni, te korisnik može odabrati jedan od prijedloga ili rezultata. Nakon što je *SearchView widget* deklariran poziva se metoda *setOnQueryTextListener()* koja postavlja sučelje za svaku radnju koju korisnik obavlja unutar samoga *widget*-a. Spomenuta metoda prima dvije javne metode tipa *boolean*. Prvi *boolean* se poziva kada korisnik pošalje željeni upit, a to se ostvaruje pritiskom tipke *Enter* na tipkovnici zaslona ili prilikom pritiska gumba za slanje upita. Drugi *boolean* se poziva kada korisnik unese odnosno izmjeni tekst upita.

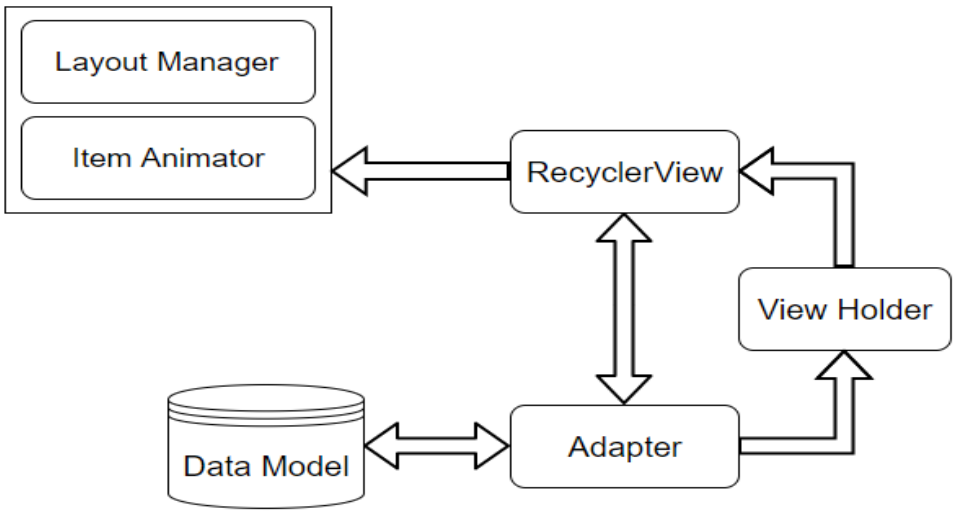
U ovom slučaju se poziva funkcija *getSearchedName()*. Unutar funkcije je najprije pozvana klasa *FoodDbHelper* koja predstavlja bazu podataka. Nakon što je pozvana metoda *getWritableDatabase()* za čitanje vrijednosti iz baze podataka, deklariran je *Cursor* koji prima vanjsku funkciju *searchItem()* koja se nalazi u klasi *FoodDbHelper*. Funkcija postavlja uvjet koji ispituje je li upisana vrijednost različita ili veća od nule, ako je uvjet zadovoljen poziva se SQL upit za pretraživanje vrijednosti. U ovom primjeru upisana vrijednost je *String searchTerm*. SQL upit će dohvatiti sve retke iz tablice *foods* gdje je stupac *food_name* jednak *String*-u *searchTerm*. Operator *LIKE* se koristi uz uvjet *WHERE* koji izdvaja one podatke koji ispunjavanju zadano stanje, a simbol % označava nula, jedan ili više znakova.

```
Cursor cursor = null;
        if (searchTerm != null && searchTerm.length() > 0) {
         String sql = "SELECT * FROM " + FoodEntry.TABLE_NAME + " WHERE " +
               FoodEntry.COLUMN_FOOD_NAME + " LIKE '%" + searchTerm + "%'";
         cursor = db.rawQuery(sql, null);
         return cursor;
```

Pritiskom na gumb Povijest otvara se zaslon koji ispisuje vremena za svaki obrok koji je korisnik spremio u bazu podataka. Ovdje korisnik odabire željeno vrijeme te pritiskom na isti otvara se skočni zaslon koji ispisuje unesenu masnoću u gramima za pojedini obrok. U slučaju kada korisnik spremi ukupne masnoće za odabrani obrok u tablicu *help*, tada se pohranjuje vrijeme unutar stupca *date*.

4.1. Razvoj aplikacije

Popis namirnica koje se nalaze u bazi se ispisuju unutar *RecyclerView*-a koji zajedno sa svojim klasama omogućuje implementaciju i dizajn dinamičkog korisničkog sučelja. U posljednje vrijeme mnoge aplikacije zahtijevaju potrebu za dizajnom koji će krajnjem korisniku pružiti prikaz više informacija i mogućnosti na jednostavan način. Upravo je zbog toga 2014. godine uveden *RecyclerView* kao rješenje za prikaz elemenata reda velikih veličina ili podataka koji se često izmjenjuju. Za potrebe ove aplikacije unutar *RecyclerView* elementa se nalaze samo osnovni *widget*-i poput *TextView* i *Button*. Prema [9], za svaki *RecyclerView* potrebno je definirati njegov vlastiti *holder* objekt nasljeđivanjem apstraktne klase *RecyclerView.ViewHolder*. Svaki *holder* ima ulogu za prikaz jednog elementa. U ovoj aplikaciji se *RecyclerView* koristi za prikaz svih namirnica u bazi podataka te za dodatne opcije (izmjena i brisanje), stoga će svaki *holder* sadržavati po jedan *TextView* element za naziv namirnice i *Button* element za izmjenu odnosno brisanje podataka. Apstraktna klasa *RecyclerView.Adapter* se koristi za upravljanje nad *holder* objektima te po potrebi kreira pogled (*engl. View*) na svaki *holder*, a svaki pogled se povezuje sa odabranim podacima. Ta veza se ostvaruje uz pomoć metode *onBindViewHolder()* i pozicije koja se dodjeljuje određenom *holder*-u. Osim spomenute metode, obavezna je još i *onCreateViewHolder()* metoda čija je uloga kreirati pogled na *holder* i pogled koji se koristi za prikaz sadržaja, najčešće XML (*engl. EXtensible Markup Language*) datoteka.



Sl. 4.1. *RecyclerView* dijagram

U primjeru je deklarirana klasa *RecyclerView.Adapter* sa svim svojim potrebnim metodama.

Unutar metode *onCreateViewHolder()* kreiran je pogled (*engl. View*) koji dohvaća XML datoteku *list_item_recycler*.

Unutar metode *onBindViewHolder()* pozvan je model *Food* u kojemu su deklarirani svi konstruktori (*engl. constructor*) i varijable nad kojima se upravlja (*getter* i *setter*). Modelu je dodijeljena lista *listFood* koja uz pomoć ključne riječi *get*, dohvaća poziciju svakog elementa koji bude naknadno definiran. Ovdje su definirani elementi *TextView* i dva *Button* elementa. Prvi *holder* dohvaća element *TextView* te postavlja ime namirnice uz pomoć naredbe *setText()* koja prima ranije spomenuti model i *getter* metodu *getName()*. Sljedeći holder dohvaća element *Button* i poziva metodu *setOnClickListener()* koja omogućuje korisniku da pritiskom unutar elementa *Button* unese izmjene nutricionističkih vrijednosti za željenu namirnicu. Ovdje je to ostvareno tako što je pozvana funkcija *editTaskDialog()* koja zapravo kreira skočni prozor u kojemu korisnik unosi izmjene za odabranu namirnicu. Treći *holder* također dohvaća element *Button* te kao i prethodni poziva metodu *setOnClickListener()*, samo što se ovaj put unutar metode poziva veza sa klasom *FoodDbHelper* u kojoj su definirane sve naredbe za upravljanje nad bazom podataka. Ovdje se poziva funkcija *deleteFood()* (objašnjena u podpoglavlju 3.3) koja briše odabranu namirnicu iz baze podataka.

```
@Override
   public FoodViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
          View view = LayoutInflater.from(parent.getContext())
                .iflate(R.layout.list_item_recycler, parent, false);
        return new FoodViewHolder(view);
    }
    @Override
    public void onBindViewHolder(FoodViewHolder holder, int position) {
        final Food singleFood = listFood.get(position);
        holder.name.setText(singleFood.getName());

        holder.editFood.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                editTaskDialog(singleFood);
            }
        });
        holder.deleteFood.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                mDatabase.deleteFood(singleFood.getId());
            }
        });

    }
```

Za unos nove namirnice u bazu podataka kreiran je *AlertDialog* koji prema [10] omogućuje prilagođeni prikaz stavki. U ovom slučaju se koriste elementi *EditText* za upis nutricionističkih vrijednosti u gramima i element *Button* koji služi za prihvaćanje upisanih vrijednosti. Nakon što je *AlertDialog* implementiran pozivaju se dvije metode *setPositiveButton()* i *setNegativeButton()*. Navedene metode se pozivaju u slučaju kada je pritisnut pozitivan, odnosno negativan gumb, a to je omogućeno uz pomoć *onClick()* metode. Unutar metode *setPositiveButton()* upisane vrijednosti u elementu *EditText* se spremaju u *double* tip podatka za nutricionističke vrijednosti, odnosno *String* tip podatka za ime namirnice, uz pomoć *getText().toString()*. Nadalje se postavlja uvjet koji provjerava ispravnost upisanih vrijednosti te ako je sve u redu poziva se funkcija *addFood()* koja uz pomoć modela *Food* i *ContentValues* vrijednosti dohvaća upisane podatke i ključnom riječi *insert* unosi te podatke u bazu podataka. Nužno je da korisnik unese ime i masnoću namirnice, a u slučaju da se neka od ostalih vrijednosti ne unese, za nju će se postaviti nula.

Kao u slučaju kada se dodaje nova namirnica u bazu podataka, na isti način je kreiran *AlertDialog* i za unos gramaže za pojedinu namirnicu. Nakon što korisnik pritisne gumb za unos gramaže poziva se metoda *setPositiveButton()* koja najprije provjerava ispravnost unesene vrijednosti. U slučaju da je uvjet zadovoljen poziva se model *GramHelp* u kojemu je kreiran konstruktor i metode *getter* i *setter* koje upravljaju sa definiranim varijablama. Najprije se poziva metoda *setter* nad varijablom *id_food* koja dohvaća id namirnice za koju korisnik unosi masu. Nadalje se *setter* metode pozivaju još dva puta nad varijablama *id_menu* i *gram*. U prvom slučaju se dohvaća varijabla *value* koja uz pomoć metode *getIntExtra()* dohvaća vrijednost koja se šalje ovisno o tome koju aktivnost je korisnik izabrao (doručak, ručak ili večera). Posljednja *setter* metoda dohvaća *gram* varijablu koja pohranjuje unesenu gramažu iz elementa *EditText* i pohranjuje je u *String*. Na kraju se sve postavljene vrijednosti pohranjuju u listu *selectedFood* pomoću ključne riječi *add*.

```
builder.setPositiveButton("Prihvati", new DialogInterface.OnClickListener() {
    @Override
    public void onClick(DialogInterface dialog, int which) {
      final double gram = Double.parseDouble(gramField.getText().toString());
      if (gram <= 0) {
        Toast.makeText(BreakfastActivity.this, " Unesi broj veći od 0",
        Toast.LENGTH_SHORT).show();
       } else {
          GramHelp gramHelp = new GramHelp();
          gramHelp.setId_food(clickedFood.getId());
          gramHelp.setId_menu(value);
          gramHelp.setGram(gram);
          selectedFood.add(gramHelp);
       }
      }
   });
```

Nakon što je korisnik odabrao namirnice za obrok i unio masu za svaku od njih, pritiskom na *FloatingActionButton* prosljeđuje se lista *selectedFood* pomoću naredbe *putExtra()* i otvara se novi zaslon koji ispisuje sve odabrane namirnice za taj obrok, a pri dnu zaslona se unutar elementa *TextView* ispisuje ukupna masnoća u gramima za odabrane namirnice. Odabrane namirnice će se ispisati uz pomoć funkcije *listTotalFood()* unutar koje se nalazi SQL upit koji će odabrati sve retke iz tablice *foods* tako što se primarni ključ tablice *foods* pridružuje sa stranim ključem *fk_foods_id* iz tablice *help*. U ovom slučaju je ta veza ostvarena uz pomoć modela *gramHelp* i *getter* metode *getId_food()*.

```
String sql = " SELECT * FROM " + FoodEntry.TABLE_NAME + " WHERE " +
        FoodEntry._ID_FOODS + " = " + gramHelp.getId_food() + ";";
```

Nakon što je deklariran element *TextView* poziva se naredba *setText()* koja prima vanjsku funkciju iz klase *FoodDbHelper* i dvije liste, *foodList* i *selectedFood*. Prva lista se dohvaća zbog toga što je unutar nje definiran model *Food* unutar kojega se nalaze sve vrijednosti o namirnici, a druga lista dohvaća podatke pohranjene u prethodnoj aktivnosti, osnosno privatne ključeve tablica *Food* i *Help* i spremljenu masu. Vanjska funkcija *multiplyFat()* najprije postavlja sigurnosni uvijet i provjerava ima li podataka u listama. Dvije *for each* petlje pretražuju liste i dohvaćaju njihove vrijednosti. Nakon pretraživanja postavlja se uvijet koji pridružuje primarni ključ iz tablice *food* sa stranim ključem *fk_foods_id* iz tablice *help*, te ako je uvijet zadovoljen množi se masnoća sa masom u gramima.

```
public String multiplyFat(List<Food> listFood, List<GramHelp> listGram) {
        double totalFat = 0;
        if (listFood.size() == 0 || listGram.size() == 0) {
            return "0";
        }
        for (Food food : listFood) {
            for (GramHelp gramHelp : listGram) {
                if (food.getId() == gramHelp.getId_food()) {
                    totalFat += food.getFat() * gramHelp.getGram();
                    break;
                }
            }
        }
        return String.valueOf(totalFat);
    }
```

Nakon što se unutar elementa *TextView* ispisala ukupna masnoća za odabrane namirnice korisnik ima mogućnost spremiti taj rezultat u tablicu *help* unutar stupca *gram_total*. Spremanje se obavlja pritiskom na *FloatingActionButton* koji je posljednji takav u ovom nizu zaslona. Prilikom spremanja ukupne masnoće, pohranjuje se i vrijeme unutar stupca *date*. Dohvaćanje vremena se obavlja pomoću apstraktne klase *DateFormat* unutar koje se nalaze mnoge podklase poput *SimpleDateFormat* koja dozvoljava pretvorbu iz nadnevka u tekst, tekst u nadnevak i normalizaciju (proces koji čini nešto normalno ili redovito). U ovom primjeru način prikaza vremena je 'mjesec dan godina | sati:minute'. Nadalje pomoću naredbe *format* obavlja se promjena trenutnog vremena u *String*. Naredba prima apstraktnu klasu *Calendar* koja nudi metode za pretvorbu između određenog trenutka u vremenu i skupa kalendarskih polja kao što su *YEAR*, *MONTH*, *DAY_OF_MONTH*, *HOUR* i tako dalje te naredbom *getTime()* vraća se objekt koji predstavlja kalendarsko vrijeme.

```
DateFormat date = new SimpleDateFormat("MMM dd yyyy | HH:mm");
String dateFormat = date.format(Calendar.getInstance().getTime());
```

Nadalje se uz pomoć *setter()* metode *String dateFormat* pohranjuje unutar modela *GramHelp*. Ukupna masnoća obroka se pohranjuje u bazu podataka pomoću funkcije *addGramTotal()* koja uz metode *getWritableDatabase()* i *ContentValues* vrijednosti pohranjuje ukupnu masnoću u gramima i vrijeme u tablicu *help*. Zajedno sa ukupnom masnoćom i vremenom također se pohranjuju i svi podaci iz liste *selectedFood()*, odnosno masa za pojedinu namirnicu i privatni ključevi tablica *food* i *menu* koji predstavljaju jednu namirnicu i jedan obrok (doručak, ručak ili večera).

Posljednji od tri gumba unutar glavnog zaslona je gumb Povijest koji otvara zaslon unutar kojega se nalazi *RecyclerView*, kao i kod prethodnih zaslona. Uz pomoć funkcije *listHistory()* ispisuju se sva vremena koja su spremljena u bazi podataka. Ispis se izvršava pomoću SQL upita koji odabire stupce *fk_menu_id*, *date* i zbroj stupca *gram_total* iz tablice *help*. Zatim ih grupira najprije po stupcu *date* te zatim po stupcu *fk_menu_id*. Na kraju se popis slaže silaznim redom prema stupcu *date*.

```
String sql = "SELECT SUM (" +FoodEntry.COLUMN_GRAM_TOTAL + ") ,"
        +FoodEntry.FOREIGN_MENU_ID + ", " +FoodEntry.COLUMN_DATE
        + " FROM " +FoodEntry.TABLE_NAME_HELPER
        + " GROUP BY " +FoodEntry.COLUMN_DATE + ", "
        +FoodEntry.FOREIGN_MENU_ID
        + " ORDER BY " +FoodEntry.COLUMN_DATE + " DESC ";
```

Također se uz prikaz vremena prikazuje obrok koji je povezan sa tim vremenom. Pritiskom na jedno od vremena otvara se skočni prozor koji ispisuje o kojemu je obroku riječ i koliko je ukupnih masnoća pohranjeno za taj obrok. Kao i u slučaju kada se ispisuju informacije o nutricionističkim vrijednostima pritiskom na neku namirnicu, tako je i u ovom slučaju. Nakon što su deklarirana dva *TextView* elementa, istima se uz pomoć metode *setText()* postavljaju određene vrijednosti koje su definirane unutar modela *GramHelp*. Nakon toga je kreiran *AlertDialog*, odnosno skočni prozor unutar kojega se za svaki *TextView* element pozivaju pohranjene vrijednosti.

5. ZAKLJUČAK

U današnje vrijeme je svakome dostupan barem neki model pametnog uređaja koji čovjeku olakšava život i štedi vrijeme. U samo nekoliko godina cijena pametnih mobilnih uređaja je pala toliko da je gotovo svakom čovjeku dostupna ta tehnologija te je život bez iste jednostavno nemoguće za pomisliti.

Prema podacima za cijelu 2016. godinu, svjetskim tržištem pametnih mobilnih uređaja definitivno dominira *Android* operacijski sustav sa 86.2% korisnika u odnosu na 12.9% iOS korisnika. Taj postotak zajedno sa tri milijuna aplikacija na *GooglePlay* trgovini čini *Android* prvim izborom gotovo svakog korisnika. Sigurno je *open source* jedan od glavnih razloga tim brojevima te dostupnost učenja i razvoju novih funkcionalnosti i tehnologija.

Prilikom izrade *Android* aplikacije usvojeno je veliko znanje i iskustvo koje će sigurno biti temelj kako za buduće projekte tako i za daljnji razvoj ove aplikacije. Na prvom mjestu će se raditi na stabilnosti i poboljšanju usluga koje aplikacija pruža. Krajnji cilj razvoja aplikacije je premjestiti bazu podataka na internet gdje će se omogućiti da svaki korisnik može sa svojom e-mail adresom kreirati vlastiti korisnički račun i koristiti aplikaciju onako kako on želi. Na taj način podaci koje korisnik sprema u aplikaciji će se moći prenijeti na bilo koji *Android* uređaj te tako nastaviti sa svojim radom.

Ideja oko daljnjeg razvoja aplikacije je stvorena kao želja za upoznavanje novih metoda pohrane podataka, upoznavanje novih tehnologija u smislu povezivanja i dijeljenja podataka sa više uređaja. Gotovo svaka aplikacija danas ima neki način dijeljenja i povezivanja na mrežu, te je dakako i to jedan od razloga za nastavak razvoja ove aplikacije.

Literatura

[1] Wikipedia, https://en.wikipedia.org/wiki/Android_(operating_system), 9.6.2017.

[2] Arhitektura, https://developer.android.com/guide/platform/index.html, 9.6.2017.

[3] SQLite, https://sqlite.org/features.html, 11.6.2017.

[4] SQLiteOpenHelper, https://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper.html,

5.7.2017

[5] ContentValues, https://developer.android.com/reference/android/content/ContentValues.html, 5.7.2017.

[6] SQLite, https://developer.android.com/training/basics/data-storage/databases.html, 29.6.2017.

[7] Intent, https://developer.android.com/reference/android/content/Intent.html, 5.7.2017

[8] SearchView, https://developer.android.com/reference/android/widget/SearchView.html, 10.9.2017.

[9] RecyclerView, https://developer.android.com/guide/topics/ui/layout/recyclerview.html, 15.8.2017.

[10] AlertDialog, https://developer.android.com/guide/topics/ui/dialogs.html, 22.8.2017.

SAŽETAK

Ovaj rad bavi se razvojem Android aplikacije za mjerenje unosa masnoće u hrani. Aplikacija je pisana pomoću programskog jezika *Java* u alatu *Android Studio*.

Aplikacija omogućuje unos namirnica za doručak, ručak i večeru te korisnik ima na izbor namirnice koje su prethodno upisane u bazi podataka ili namirnice koje je sam unio u bazu podataka. Uz svaku namirnicu pohranjene su nutricionističke vrijednosti u gramima. Korisnik ima na izbor dodavanje novih ili izmjenjivati i brisati namirnice iz baze podataka. Unosom mase za odabrane namirnice izračunava se ukupna masnoća cijeloga obroka u gramima. Zbroj se sprema u bazu podataka zajedno sa svim ostalim vrijednostima.

Kao krajnji rezultat ove aplikacije omogućeno je lakše praćenje unosa masnoće u tijelo posebno kod osoba sa poremećajem hiperlipidemija. Aplikacija na jednom mjestu sadrži sve nutricionističke vrijednosti jedne namirnice te se vrlo brzo i jednostavno dolazi do rezultata koji bi inače zahtijevao više truda i vremena.

Ključne riječi: *Android, Java, SQLite,* baza podataka

ABSTRACT

Application for Android platform for measuring fat in food

This paper is about development of an Android application for measuring fat intake in food. The application was written using Java programming language with Androd Studio tool.

The application allows breakfast, lunch and dinner food intake and the user has the choice of food that was previously entered into the database or food that the user has entered into the database. Nutrition values are stored in grams with each food. The user has the choice of adding new, or editing and deleting food items from the database. Entering weight for selected foods, application calculates the total fat of the whole meal in grams. The sum is saved in the database together with all other values.

As an final result of this application, it is easier to track fat intake in the body, especially for people with hyperlipidemia. The application contains all nutrition values of a food item in one place and it quickly and simply comes up with result that would otherwise require more effort and time.

Key words: *Android*, *Java*, *SQLite*, database

ŽIVOTOPIS

Stjepan Salopek rođen je 20. srpnja 1994. godine u Bjelovaru. Nakon završetka osnovne škole u Siraču, 2008. godine upisuje Tehničku školu u Daruvaru, smjer tehničar za računalstvo. Položenom državnom maturom, 2013. godine upisuje Elektrotehnički fakultet u Osijeku, stručni studij elektrotehnike, smjer informatika.

Stjepan Salopek

_____