Mladi za napredek Maribora 2018
35.srečanje

Android aplikacija Bright Future

Inovacijsko področje: računalništvo in informatika
Inovacijski predlog

Avtor: MARIO MOČNIK, PASCAL VOGRIN
Mentor: JOŽE ŠTRUCL
Šola: SREDNJA ELEKTRO-RAČUNALNIŠKA ŠOLA MARIBOR

Maribor, 2018

Mladi za napredek Maribora 2018
35.srečanje

Android aplikacija Bright Future

Inovacijsko področje: računalništvo in informatika
Inovacijski predlog

Maribor, 2018

KAZALO VSEBINE

POVZETEK *5*

UVOD *6*

DELOVANJE APLIKACIJE *7*

OPIS **7**

POSTOPEK DELA **7**

Koncept 7

Podatkovna baza 8

PHP Strežnik 10

Aplikacija 12

Obrazložitev pridobitve podatkov iz HTML kode 29

UPRAVLJANJE APLIKACIJE *32*

NADALJNI NAČRT RAZVIJANJA APLIKACIJE *35*

Uporabniški vmesnik **35**

Optimizacija aplikacije **36**

DRUŽBENA ODGOVORNOST *37*

ZAKLJUČEK *38*

KAZALO SLIK

S0: Grafični potek komunikacije med podatkovno bazo in telefonsko aplikacijo 7
S1: Prikaz relacij in tabel podatkovne baze 8
S2: Prikaz podatkov vnesenih v podatkovno bazo 8
S3: Prikaz SQL kode, ki bo uporabljena za ustvarjanje podatkovne baze na strežniku 9
S4: Prikaz PHP kode, ki je na strežniku 10
S5: Prikaz HTML strani, ko je iskana tabela »Šola« 10
S6: Prikaz podatkov v HTML kodi 11
S7: Začetne spremenljivke 12
S8: Metoda, ki se kliče ob zagonu aplikacije 12
S9: Razred, ki je odgovoren za podatkovno bazo na telefonski napravi 12
S10: Metode za dodajanje podatkov v podatkovno bazo 13
S11: Metodi za pridobivanje podatkov s pomočjo iskanega niza 13
S12: Metoda odgovorna za pridobivanje podatkov glede na iskan niz, ki ga poda uporabnik 14
S13: Metoda odgovorna za ugotavljanje ali ima telefon dostop do internetne povezave 15
S14: Primer uporabe metode za preverjanje internetne povezave v android aplikaciji 15
S15: Metodi, ki upravljata posodabljanje podatkovne baze 16
S16: Metoda odgovorna za zagon pridobitve podatkov 16
S17: Spremenljivke v razredu odgovornem za pridobivanje podatkov 17
S18: Konstruktor in metoda odgovorni za pridobitev podatkov ter klicanje metode za obdelave podatkov 17
S19: Metoda odgovorna za pridobitev podatkov iz HTML kode in vračanje teh podatkov 18
S20: Prikaz for zanke, ki je odgovorna za shranjevanje podatkov v lokalno podatkovno bazo 18
S21: Prikaz razreda odgovornega za shranjevanje podatkov tipa »Sola« 19
S22: Prikaz kode, ki shranjuje podatke v lokalno podatkovno bazo 20
S23: Koda, ki kliče metode iz razreda odgovornega za upravljanje z podatkovno bazo 21
S24: Prikaz pošiljanje podatka 21
S25: Handler odgovoren za pridobivanje podatkov iz podatkovne baze 21
S26: Metoda, ki kliče metodo odgovorno za pridobivanje podatkov 22
S27: Poslušalec odgovoren za posodabljanje prikaz podatkov ob spremembi iskanega niza 22
S28: Izgled aplikacije in iskanega polja 23
S29: Metoda odgovorna za iskanje podakotv in njihovo prikazovanje glede na iskan niz, ki ga je vnesel uporabnik 23
S30: Metodi odgovorni za ustvarjanje vidnih rezultatov 24
S31: Vidni rezultati v aplikaciji glede na iskan niz 25
S32: Izgled aplikacije ob kliku na rezultat šole 25
S33: Metodi odgovorni za prikaz novih podatkov glede na šolo, ki je bila podana ob kliku 26
S34: Metodi za prikaz novih podatkov glede na stroko, ki je bila podana ob kliku 27
S35: Prikaz rezultatov v aplikaciji 28
S36: Potek pridobitve podatkov 1 29
S37: Potek pridobitve podatkov 2 30
S38: Grafični prikaz načina shranjevanja podatkov 31
S39: Pogoj za obdelavo podatkov 31
S40: Aplikacija ob zagonu 32
S41: Aplikacija ob iskanju šole ali stroke 33
S42: Aplikacija ob pritisku na rezultat tipa šole 33
S43: Aplikacija ob pritisku na rezultat tipa stroke 34
S44: Aplikacija ob ponovnem pritisku na rezultat 34
S45: Trenutno načrtovan uporabniški vmesnik 35

POVZETEK

Android aplikacija »Bright Future« omogoča uporabnikom hiter vpogled v trenutno stanje vpisnih mest srednjih šol, njihovih strokovnih izobraževalnih smereh in podatke o samih šolah. Pri tem aplikacija omogoča tudi iskanje šol s določenimi strokovnimi smeri. Sama aplikacija je bila načrtovana v programu »Android Studio«, podatkovna baza pa ustvarjena s pomočjo »Microsoft Access«. Za delovanje aplikacije je bil potreben tudi spletni strežnik, ki pa je brezplačen s strani »000webhosting«.

UVOD

Android aplikacija »Bright Future« je namenjena osnovnošolcem in srednješolcem, katere zanimajo podatki o srednjih šolah in njihovih strokovnih smeri. Sama aplikacija Je bila zasnovana v programskem okolju »Android Studio« in napisano v programskem jeziku »Java« ter »xml«. Za samo delovanje aplikacije pa je bila potrebna podatkovna baza, katera je zaradi varnosti uporabljala spletni strežnik kot vmesnik med android aplikacijo in podatkovno bazo. V času razvijanja je bilo vprašanje kako dostopati do podatkov iz podatkovne baze in jih shraniti v telefonsko napravo ampak to se je pozneje rešilo s PHP strežnikom, ki je podatke iz baze prikazal v HTML tekstu, katerega android aplikacija pozneje pretvori v podatke in jih shrani. Največji problem pri samem izdelovanju aplikacije je bila pretvorba HTML kode v podatke, ki jih je bilo mogoče shranit. Rešitev tega problem ja prikazana pozneje v poglavju »Aplikacija«. Aplikacija je v času razvijanja že uporabljala podatkovno bazo iz spletnega strežnika, ki je bil vzpostavljen pred samim začetkom dela na aplikaciji. V poteku razvijanja aplikacije je PHP strežnik opravil svoje delo in je naredil točno to, kaj je bil namenjen naredit.

DELOVANJE APLIKACIJE

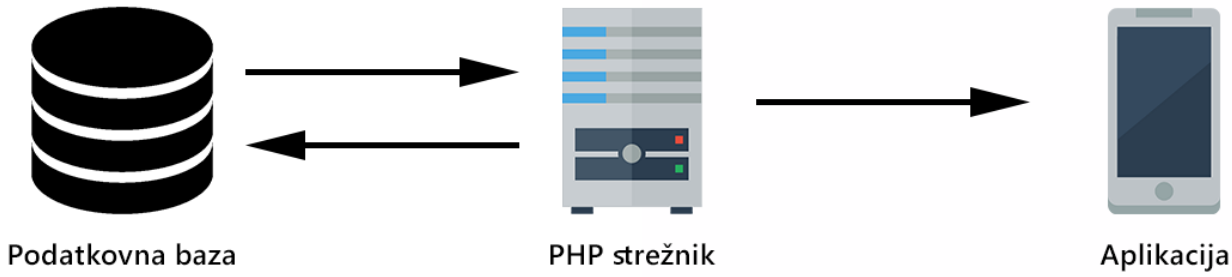
OPIS

Preizkusna različica je bila namenjena temu, da se preizkusi delovanje aplikacije in se v poznejših različicah izognit napak, ki bi se pokazale v preizkusni različici. Za začetek je bilo potrebno ugotoviti kakšno orodje uporabit za razvijanje aplikacije. Na koncu je bila odločitev »Android studio«, saj je na videz izgledalo dokaj enostavno orodje za uporabo razvijanja aplikacije.

POSTOPEK DELA

Koncept

Pred začetkom dela si je bilo potrebno najprej zamisliti kako bo aplikacija delovala. Za podatke je bilo jasno iz samega začetka, da bo potrebno uporabljati podatkovno bazo zaradi enostavnejšega posodabljanja in iskanja podatkov. Aplikacija bi zato dostopala do te podatkovne baze, ko je to možno. Če pa to ni možno pa bi aplikacija mogla imeti kopijo nazadnje shranjenih podatkov s katerimi bi lahko upravljala. Grafični prikaz dostopa do podatkov si je mogoče predstavljati s naslednjo grafično sliko:

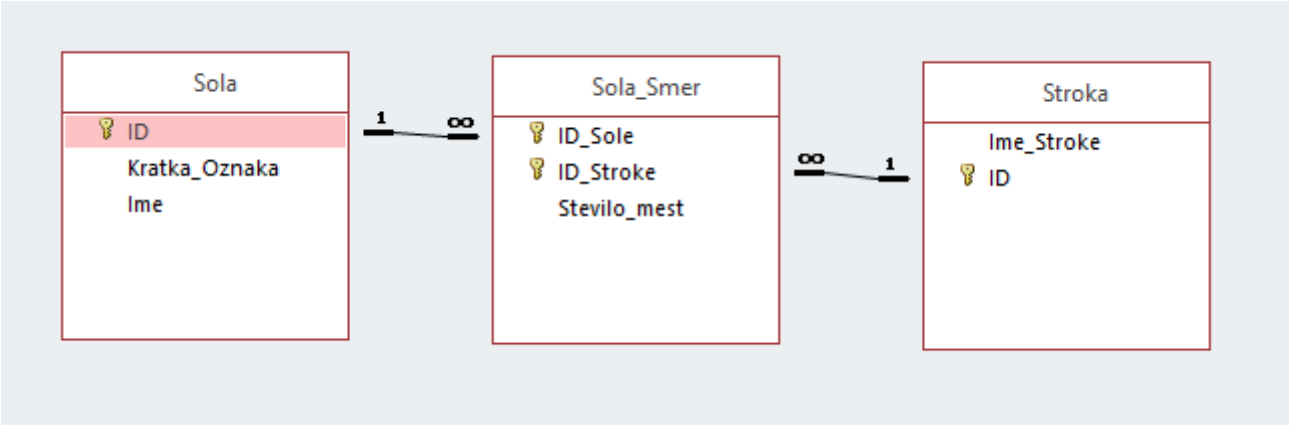


S0: Grafični potek komunikacije med podatkovno bazo in telefonsko aplikacijo

Podatke bi aplikacija pridobila tako, da bi se povezala na PHP strežnik, ki ima dostop do podatkovne baze, ta pa bi nazaj aplikaciji poslal podatke. Razlog za to je, da vpisne informacije o bazi ne bi bile v sami kode aplikacije, saj bi to omogočalo, da bi nekdo lahko dostopal do teh informacij, če bi prišel do izvorne kode aplikacije.

Podatkovna baza

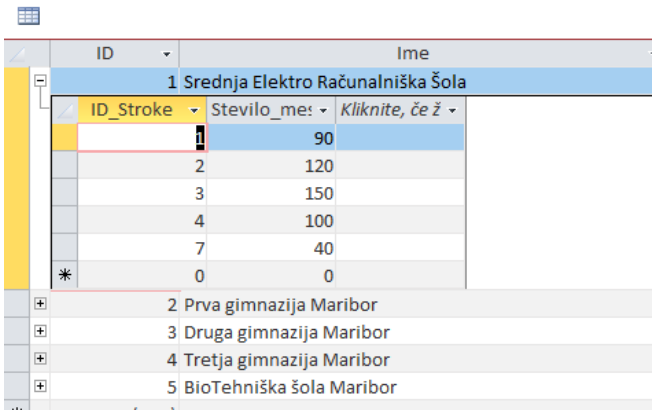
Podatkovna baza bi za osnovno prvo različico potrebovala informacije o šolah in strokovnih smereh, ki so v teh šolah. Podatkovna baza je zaradi tega razloga dobila naslednji izgled:



S1: Prikaz relacij in tabel podatkovne baze

Tabela s podatki o šolah in tabela s podatki o strokah sta bili ločeni in skupaj povezani preko nove tabele, ki ima podatke o tem, v katerih šolah je katera stroka in koliko število prostih mest ima ta stroka v tej šoli.

Ta struktura omogoča lažjo nadgradnjo v poznejših različicah baze in aplikacije. Za začetek razvijanja aplikacije se je v bazo podatkov shranilo samo malo število podatkov za testiranje.



S2: Prikaz podatkov vnesenih v podatkovno bazo

Za prenašanje podatkovne baze na strežnika se je uporabil »MS Access to MySQL«, ki je pretvori podatkovno bazo v SQL kodo in jo shranil na računalnik.

```
#
# Table structure for table 'Sola'
#

DROP TABLE IF EXISTS `Sola`;

CREATE TABLE `Sola` (
   `ID` INTEGER NOT NULL AUTO_INCREMENT,
   `Kratka_Oznaka` VARCHAR(25),
   `Ime` VARCHAR(255),
   PRIMARY KEY (`ID`)
) ENGINE=myisam DEFAULT CHARSET=utf8;

SET autocommit=1;

#
# Dumping data for table 'Sola'
#

INSERT INTO `Sola` (`ID`, `Kratka_Oznaka`, `Ime`) VALUES (1, 'SERS', 'Srednja Elektro Računalniška Šola');
INSERT INTO `Sola` (`ID`, `Kratka_Oznaka`, `Ime`) VALUES (2, 'PGM', 'Prva gimnazija Maribor');
INSERT INTO `Sola` (`ID`, `Kratka_Oznaka`, `Ime`) VALUES (3, 'DGM', 'Druga gimnazija Maribor');
INSERT INTO `Sola` (`ID`, `Kratka_Oznaka`, `Ime`) VALUES (4, 'TGM', 'Tretja gimnazija Maribor');
INSERT INTO `Sola` (`ID`, `Kratka_Oznaka`, `Ime`) VALUES (5, 'BTŠM', 'BioTehniška šola Maribor');
# 5 records
```

S3: Prikaz SQL kode, ki bo uporabljena za ustvarjanje podatkovne baze na strežniku

Ta datoteka je bila pozneje prenesena na PhPMyAdmin podatkovno bazo in pozneje uporabljena za delovanje aplikacije.

PHP Strežnik

PHP Strežnik služi kot vmesnik med aplikacijo in podatkovno bazo. Strežnik bi ob povezavi potreboval ime tabele iz katere naj pridobi podatke, te pa bi potem izpisal v HTML kodi, ki bi pa jih aplikacija shranila v lokalno bazo shranjeno na telefonu.

```
    function DobiPodatke($Tabela)
{

    $sqlget = "SELECT * FROM " . $Tabela;
    ZazeniQuery($Tabela, $sqlget);
}


    function ZazeniQuery($Ime, $Query){
    $sqldata = mysqli_query($GLOBALS['dbcon'], $Query) or die('<error>error with fetching data</error>');
        ECHO '<' . $Ime . '>';
        while($row = mysqli_fetch_array($sqldata, MYSQLI_ASSOC)){
            switch($Ime){
                case 'Sola':
                    ECHO '<vnos><Kratka_Oznaka>' . $row['Kratka_Oznaka'] . '</Kratka_Oznaka><Ime>' . $row['Ime
                    break;
                case 'Stroka':
                    ECHO '<vnos><Ime_Stroke>' . $row['Ime_Stroke'] . '</Ime_Stroke></vnos>';
                    break;
                case 'Sola_Smer':
                    ECHO '<vnos><ID_Sole>' . $row['ID_Sole'] . '</ID_Sole><ID_Stroke>' . $row['ID_Stroke'] . '
                    break;
            }
        }
        ECHO '</' . $Ime . '>';
    }
```

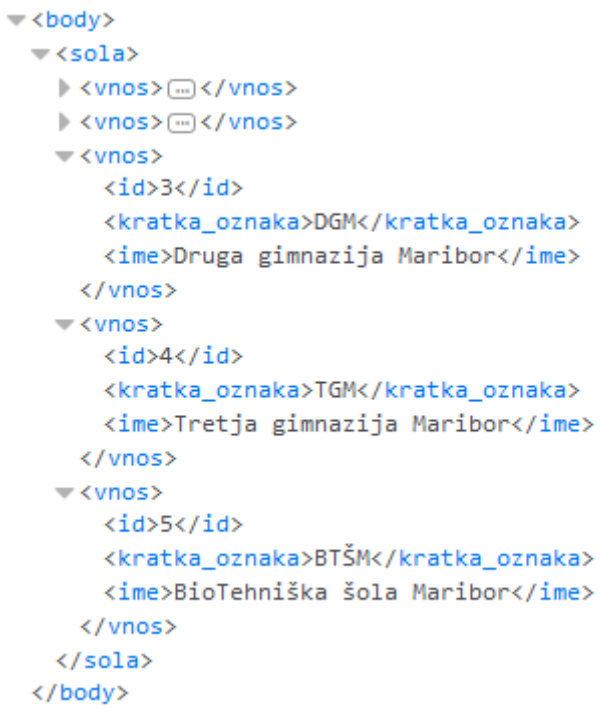
S4: Prikaz PHP kode, ki je na strežniku

Kot rezultat te PHP kode se ob iskanju nad tabelo »Sola« pridobi naslednji reultat:

1SERSSrednja Elektro Računalniška Šola2PGMPrva gimnazija Maribor3DGMDruga gimnazija Maribor4TGMTretja gimnazija Maribor5BTŠMBioTehniška šola Maribor

S5: Prikaz HTML strani, ko je iskana tabela »Šola«

Človeku je to težko berljivo, saj med podatki ni vidnih razmikov. Ampak android aplikaciji je ta HTML koda berljiva saj so različni podatki ločeni s HTML značkami, kot prikazani tukaj:



S6: Prikaz podatkov v HTML kodi

S pomočjo tega je nato lažje pridobit podatke iz HTML kode in jih tudi shranit. Za samo prenašanje te datoteke na strežnik se je uporabil program »FileZila«.

Aplikacija

Ob zagonu aplikacije se nastavijo deklaracije nekaterih razredov, ki so potrebni za delovanje aplikacije.

```
protected Handler PutIntoSQL = new Handler(new HandleDBClass());
protected static DBHandler DBEntryHandler;
private LinearLayout DisplayLayout;

protected static LinearLayout.LayoutParams GlobalTextDisplayParam = new LinearLayout.LayoutParams(LinearLayout.LayoutParams.WRAP_CONTENT, LinearLayout.LayoutParams.WRAP_CONTENT);
```

S7: Začetne spremenljivke

Nastavijo se tudi parametri za razporeditev rezultatov, ki bodo pozneje prikazani v sami aplikaciji. Nastavijo se tudi spremenljivke, ki so bile deklariranje odzgoraj. Ena izmed spremenljivk , ki pa se uporablja skoraj skozi celotno aplikacijo pa je spremenljivka DBEntryHandler, ki pa je instanca razreda DBHandler, ki služi, kot upravitelj podatkovne baze.

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    GlobalTextDisplayParam.setMargins( left: 20,  top: 20,  right: 20,  bottom: 0);
    GlobalTextDisplayParam.gravity = Gravity.START;


    EditText TableSelectionText = findViewById(R.id.DBSelection);    //find main linear view to put the text into
    DisplayLayout = findViewById(R.id.DisplayLayout);

    DBEntryHandler = new DBHandler(getBaseContext(),  name: null,  factory: null,  version: 1);
```

S8: Metoda, ki se kliče ob zagonu aplikacije

Sama baza potrebuje samo par stavkov za tvorbo vendar več stavkov za upravljanje.

```
private static final String DATABASE_NAME = "DBSaved.db";
private static final String TABLE_SOLA = "Sola";
private static final String TABLE_SOLA_SMER = "Sola_Smer";
private static final String TABLE_STROKA = "Stroka";



public DBHandler(Context context, String name, SQLiteDatabase.CursorFactory factory, int version) {
    super(context, DATABASE_NAME, factory, version);
}

@Override
public void onCreate(SQLiteDatabase sqLiteDatabase) {
    String querySola = "CREATE TABLE " + TABLE_SOLA + "( ID INTEGER PRIMARY KEY ," + "Kratka_Oznaka TEXT ," + "Ime TEXT );";
    String queryStroka = "CREATE TABLE " + TABLE_STROKA + "( ID INTEGER PRIMARY KEY, " + "Ime_Stroke TEXT );";
    String querySola_Smer = "CREATE TABLE " + TABLE_SOLA_SMER + "( ID_Sole INTEGER, " + "ID_Stroke INTEGER, " + "Stevilo_mest INTEGER );";
    sqLiteDatabase.execSQL(querySola);
    sqLiteDatabase.execSQL(queryStroka);
    sqLiteDatabase.execSQL(querySola_Smer);

}

@Override
public void onUpgrade(SQLiteDatabase sqLiteDatabase, int i, int il) {
    sqLiteDatabase.execSQL("DROP TABLE IF EXISTS " + TABLE_SOLA_SMER);
    sqLiteDatabase.execSQL("DROP TABLE IF EXISTS " + TABLE_STROKA);
    sqLiteDatabase.execSQL("DROP TABLE IF EXISTS " + TABLE_SOLA);
    onCreate(sqLiteDatabase);
}
```

S9: Razred, ki je odgovoren za podatkovno bazo na telefonski napravi

Zato so ustvarjene posebne metode za pridobivanje in ustvarjanje podatkov v sami bazi.

```
void addSola(Sola sola){
    ContentValues values = new ContentValues();
    values.put("ID", sola.get_ID());
    values.put("Kratka_Oznaka", sola.get_Kratka_Oznaka());
    values.put("Ime", sola.get_Ime());
    SQLiteDatabase db = getWritableDatabase();
    db.insert(TABLE_SOLA, nullColumnHack: null , values);
    db.close();
}

void addStroka(Stroka stroka){
    ContentValues values = new ContentValues();
    values.put("ID", stroka.get_ID());
    values.put("Ime_Stroke", stroka.get_Ime_Stroke());
    SQLiteDatabase db = getWritableDatabase();
    db.insert(TABLE_STROKA, nullColumnHack: null , values);
    db.close();
}

void addSolaSmer(Sola_Smer sola_smer){
    ContentValues values = new ContentValues();
    values.put("ID_Sole", sola_smer.get_ID_Sole());
    values.put("ID_Stroke", sola_smer.get_ID_Stroke());
    values.put("Stevilo_mest", sola_smer.get_Stevilo_mest());
    SQLiteDatabase db = getWritableDatabase();
    db.insert(TABLE_SOLA_SMER, nullColumnHack: null , values);
    db.close();
}
```

S10: Metode za dodajanje podatkov v podatkovno bazo

Za pridobivanje podatkov so ustvarjene tri metode, dve od teh pa sta uporabljeni za pridobivanje podatkov, če je šola ali stroka znan podatek ob iskanju.

```
Cursor getStrokaFromSchool(String SolaIme){
    SQLiteDatabase db = this.getWritableDatabase();

    String query = "SELECT " + TABLE_STROKA + ".Ime_Stroke, " + TABLE_SOLA_SMER + ".Stevilo_mest FROM (("
    Cursor data = db.rawQuery(query, selectionArgs: null);

    return data;
}

Cursor getSchoolFromStroka(String StrokaIme){
    SQLiteDatabase db = this.getWritableDatabase();

    String query = "SELECT " + TABLE_SOLA + ".Ime, " + TABLE_SOLA_SMER + ".Stevilo_mest FROM ((" + TABLE_
    Cursor data = db.rawQuery(query, selectionArgs: null);

    return data;
}
```

S11: Metodi za pridobivanje podatkov s pomočjo iskanega niza

Zadnja metoda za iskanje podatkov pa se uporablja, ko uporabnik išče za šole in podatke na glavnem zaslonu aplikacije. Ta metoda vrne vse šole in stroke, katerih ime se vsaj delno ujema z iskanim tekstom, ki ga je vnesel uporabnik.

```
Cursor customQuery(String table, String query, String conditions){
    SQLiteDatabase db = this.getWritableDatabase();
    query = "SELECT " + query + " FROM ";
    switch (table){
        case TABLE_SOLA: query+=TABLE_SOLA;
            break;
        case TABLE_STROKA: query+=TABLE_STROKA;
            break;
        case TABLE_SOLA_SMER: query+=TABLE_SOLA_SMER;
            break;
    }

    if(conditions!=null){
        query+=" WHERE " + conditions;
    }


    Cursor data = db.rawQuery(query,  selectionArgs: null);

    return data;
}
```

S12: Metoda odgovorna za pridobivanje podatkov glede na iskan niz, ki ga poda uporabnik

Preden lahko aplikacija pridobi podatke s katerimi bo delala mora preverit, če ima dostop do interneta. To omogoča naslednja metoda, ki je poklicana vedno, ko aplikacija hoče pridobit podatke s PHP

strežnika.

```
private boolean internetConnectionAvailable(int timeOut) {
    InetAddress inetAddress[] = new InetAddress[2];

    try {
        Future<InetAddress> interfaceSite = Executors.newSingleThreadExecutor().submit(new Callable<InetAddress>() {
            @Override
            public InetAddress call() {
                try {
                    return InetAddress.getByName("illegible-deflector.000webhostapp.com");
                } catch (UnknownHostException e) {
                    return null;
                }
            }
        });

        Future<InetAddress> interfaceDB = Executors.newSingleThreadExecutor().submit(new Callable<InetAddress>() {
            @Override
            public InetAddress call() {
                try {
                    return InetAddress.getByName("brightfuturefront.000webhostapp.com");
                } catch (UnknownHostException e) {
                    return null;
                }
            }
        });
        //Log.i("LOG", "Checking interface site status");
        inetAddress[0] = interfaceSite.get(timeOut, TimeUnit.MILLISECONDS);
        interfaceSite.cancel( b: true);

        //Log.i("LOG", "Checking DB site status");
        inetAddress[1] = interfaceDB.get(timeOut, TimeUnit.MILLISECONDS);
        interfaceDB.cancel( b: true);

    } catch (Exception e) {
        //Log.i("LOG", "Error " + e);
    }

    if(inetAddress[0] != null  && inetAddress[1] != null) return true;
    return false;
}
```

S13: Metoda odgovorna za ugotavljanje ali ima telefon dostop do internetne povezave

Metoda preveri povezljivost do dveh spletnih mest na način, da poskusi pridobit IP naslov teh spletnih mest. Če IP naslovov ne more razločit pomeni, da ali aplikacija nima vklopljenih mobilnih podatkov ali brezžične povezave ali pa, da je naprava povezana na brezžično povezavo vendar od te nima dostopa do interneta.

Aplikacija uporabi to funkcijo, ko se odloča ali bi podatkovno bazo ob zagonu posodobila ali ne.

```
if (internetConnectionAvailable( timeOut: 2000)) {
    emptyDatabase();
    createDatabase();
}
```

S14: Primer uporabe metode za preverjanje internetne povezave v android aplikaciji

```
protected void createDatabase() {
    //Log.i("LOG", "Create database called");

    RunDBData( who: 0);

}


protected void emptyDatabase() {
    //Log.i("LOG", "Empty database called");

    try {
        DBEntryHandler.close();
        if (getBaseContext().deleteDatabase( s: "DBSaved.db")) {
            Toast.makeText(getBaseContext(), text: "Databse Deleted", Toast.LENGTH_LONG).show();
        }
    } catch (Exception e) {
        //Log.i("LOG", "Error deleting databse");
    }
}
```

S15: Metodi, ki upravljata posodabljanje podatkovne baze

Če ima aplikacija dostop do interneta se bo prvo lokalna podatkovna baza shranjena na telefonu izbrisala in nato ponovno ustvarila in napolnila z podatki, ki jih pridobi iz PHP strežnika. Podatke pridobi z naslednjo metodo.

```
private void RunDBData(final int who) {
    final String[] DBTables = {"Sola", "Stroka", "Sola_Smer"};
    if (who < 3) {
        Runnable DBRun = new Runnable() {
            @Override
            public void run() {
                new TextExtractor(DBTables[who], getBaseContext(), PutIntoSQL, who);
            }
        };

        Thread DBThread = new Thread(DBRun);
        DBThread.start();
    }
```

S16: Metoda odgovorna za zagon pridobitve podatkov

Ta funkcija ustvari novo instanco razreda TextExtractor, ki v podatkovni bazi napolni eno tabelo. Ker je podatkov lahko veliko, se ta koda izvede v novi niti s pomočjo Runnable razreda. Sam razred, ki je odgovoren za pretvarjanje HTML kode v podatke, pa je razred TextExtractor. V sami kodi ima zapisan naslov, do katerega se more povezat za podatke.

```
class TextExtractor {
    private  final String DBLink = "https://illegible-deflector.000webhostapp.com/Site3/dbtest.php?Tabela=";
    private ArrayList<HashMap> Entries;
    private static BoyerMooreHorspoolRaita SearchUnit = new BoyerMooreHorspoolRaita();
    private  String QueryName;
    private Handler DBHandler;
    private int iAm;
```

S17: Spremenljivke v razredu odgovornem za pridobivanje podatkov

Razred vsebuje tudi posebni razred BoyerMooreHorspoolRaita, ki je del StringSearch Razreda. Ta razred omogoča hitro iskanje delov teksta v nizu in se uporablja za iskanje napak, ki jih je javil strežnik , v HTML kodi.

```
TextExtractor(String GetTable, Context context, Handler DBHandler, int iAm){
    this.DBHandler = DBHandler;
    this.iAm = iAm;
    String LinkParser = DBLink + GetTable;
    QueryName = GetTable;
    ParseDBFront(LinkParser, context);
    //Log.i("LOG", "TextExtractor working " + Working);
}


private void ParseDBFront(final String link, final Context context) {
    Ion.with(context).load(link).asString().setCallback((e, result) -> {
            ////Log.i("LOG", "Log: " + result);
            Entries = dataHandler(result);
            SendToDataBase();

            Message msg = Message.obtain();
            msg.obj = iAm;
            DBHandler.sendMessage(msg);
    });
}
```

S18: Konstruktor in metoda odgovorni za pridobitev podatkov ter klicanje metode za obdelave podatkov

TextExtractor takoj ob svoji ustvarjenost nastavi vse potrebne podatke, ki jih je dobil v konstruktoru in nato pokliče metodo ParseDBFront, ki s pomočjo Ion Razreda ustvari asinhrono funkcijo in ob uspešni pridobitvi podatkov nastavi vrednost spremenljivke Entries na vsebino podatkov, ki jih je metoda dataHandler pridobila iz HTML kode.

```
private ArrayList<HashMap> dataHandler(String HTMLCode){
    ArrayList<HashMap> EntriesDB = new ArrayList<>();

    int checkHTML = SearchUnit.searchString(HTMLCode,  pattern: "<error>");
    int checkHTML2 = SearchUnit.searchString(HTMLCode, QueryName);
    if(checkHTML<0 && checkHTML2>=0) {
        int startPos = QueryName.length() + 2;  //consider the lenght of < > brackets
        int endPos = HTMLCode.length() - QueryName.length() - 3;
        HTMLCode = HTMLCode.substring(startPos, endPos);

        while (!HTMLCode.equals("") && HTMLCode.length() >= 12) {       //12 to check if <vnos> </vnos> exists
            HashMap<String, String> PodatkovniPar = new HashMap<>();
            PodatkovniPar.clear();
            int InfoStart = 6;
            int InfoEnd = SearchUnit.searchString(HTMLCode,  pattern: "</vnos>");
            String data = HTMLCode.substring(InfoStart, InfoEnd);

            while (!data.equals("")) {
                int TableStart = SearchUnit.searchString(data,  pattern: ">");
                String Table1 = data.substring(1, TableStart);
                data = data.substring(TableStart + 1);
                int TableEnd = SearchUnit.searchString(data, Table1) - 2;     // consider </
                String DataHold = data.substring(0, TableEnd);
                data = data.substring(TableEnd + Table1.length() + 3);
                PodatkovniPar.put(Table1, DataHold);
            }

            HTMLCode = HTMLCode.substring(InfoEnd + 7);
            EntriesDB.add(PodatkovniPar);
        }
    }
    else{
        EntriesDB=null;
    }

    return EntriesDB;
}
```

S19: Metoda odgovorna za pridobitev podatkov iz HTML kode in vračanje teh podatkov

Ob uspešni pridobitvni podatkov se zažene metoda SendToDataBase, ki vse podatke, ki so bili pridobljeni, shrani v bazo podatkov na telefonu.

```
private void SendToDataBase(){

    //Log.i("LOG", "Logged completion of : " + QueryName + " - ");

    for(HashMap<String, String> var : Entries){

        Sola DBEntryDataRowSola = new Sola();
        Stroka DBEntryDataRowStroka = new Stroka();
        Sola_Smer DBEntryDataRowSola_Smer = new Sola_Smer();


        for(HashMap.Entry val : var.entrySet()){
            //Log.i("LOG", "name - " + val.getKey() + " - Value : " + val.getValue());
```

S20: Prikaz for zanke, ki je odgovorna za shranjevanje podatkov v lokalno podatkovno bazo

To naredi s tem, da ustvari tri nove instance razredov Sola, Stroka in Sola_Smer, ki služijo, kot ena vrstica podatk v

```
public class Sola {
    private int _ID;
    private String _Kratka_Oznaka;
    private String _Ime;

    public Sola(){

    }

    public Sola(int ID, String Kratka_Oznaka, String Ime){
        _ID = ID;
        _Kratka_Oznaka = Kratka_Oznaka;
        _Ime = Ime;
    }


    public int get_ID() { return _ID; }

    public void set_ID(int _ID) {
        this._ID = _ID;
    }

    public String get_Kratka_Oznaka() { return _Kratka_Oznaka; }

    public void set_Kratka_Oznaka(String _Kratka_Oznaka) { this._Kratka_Oznaka = _Kratka_Oznaka; }

    public String get_Ime() { return _Ime; }

    public void set_Ime(String _Ime) { this._Ime = _Ime; }
}
```

S21: Prikaz razreda odgovornega za shranjevanje podatkov tipa »Sola«

V nadaljevanju pa različni switch stavki dodajajo podatke v te instance razredov, glede na ime, ki je bila podana ob ustvarjenju instance razreda TextExtractor.

```
try {
    switch (QueryName) {

        case "Sola": {
            switch (val.getKey().toString()) {
                case "ID":
                    DBEntryDataRowSola.set_ID(Integer.parseInt(val.getValue().toString()));
                    break;
                case "Ime":
                    DBEntryDataRowSola.set_Ime(val.getValue().toString());
                    break;
                case "Kratka_Oznaka":
                    DBEntryDataRowSola.set_Kratka_Oznaka(val.getValue().toString());
                    break;
            }
            break;
        }

        case "Stroka": {
            switch (val.getKey().toString()) {
                case "ID":
                    DBEntryDataRowStroka.set_ID(Integer.parseInt(val.getValue().toString()));
                    break;
                case "Ime_Stroke":
                    DBEntryDataRowStroka.set_Ime_Stroke(val.getValue().toString());
                    break;
            }
            break;
        }
```

S22: Prikaz kode, ki shranjuje podatke v lokalno podatkovno bazo

Naslednji switch stavek doda podatke, ki so bili nastavljeni s prejšnjimi switch stavki. Doda jih pa s tem, da kliče metodo, ki je bila ustvarjena za vse tri tabele.

```
    switch (QueryName){
        case "Sola": MainActivity.DBEntryHandler.addSola(DBEntryDataRowSola);
            //Log.i("LOG", "Added new Sola Entry to database!");
            break;
        case "Stroka": MainActivity.DBEntryHandler.addStroka(DBEntryDataRowStroka);
            //Log.i("LOG", "Added new Stroka Entry to database!");
            break;
        case "Sola_Smer": MainActivity.DBEntryHandler.addSolaSmer(DBEntryDataRowSola_Smer);
            //Log.i("LOG", "Added new Sola_Smer Entry to database!");
            break;
    }
}
```

S23: Koda, ki kliče metode iz razreda odgovornega za upravljanje z podatkovno bazo

Ob končanem vnašanju podatkov za trenutno tabelo se ustvari novi objekt Message, ki služi kot sporočilo, ki se bo poslalo Handlerju, ki je bil ustvarjen na začetku programa. V to sporočilo vneše podatek iAm, ki služi kot identifikator tabele, katere podatke je pridobil ta instance razreda TextExtractor.

```
Message msg = Message.obtain();
msg.obj = iAm;
DBHandler.sendMessage(msg);
```

S24: Prikaz pošiljanje podatka

Sam Handler pa je samo uporabljen za zagon že obstoječe metode RunDBData, vendar z drugačnim indentifikatorjem.

```
class HandleDBClass implements Handler.Callback {


    @Override
    public boolean handleMessage(Message message) {
        int HeIS = Integer.valueOf(message.obj.toString());
        //Log.i("LOG", "I am : " + HeIS);
        RunDBData( who: HeIS + 1);

        return true;
    }
}
```

S25: Handler odgovoren za pridobivanje podatkov iz podatkovne baze

To se ponovi samo tri-krat, saj ima podatkovna baza samo tri tabele, katerih podatke mora pridobit.

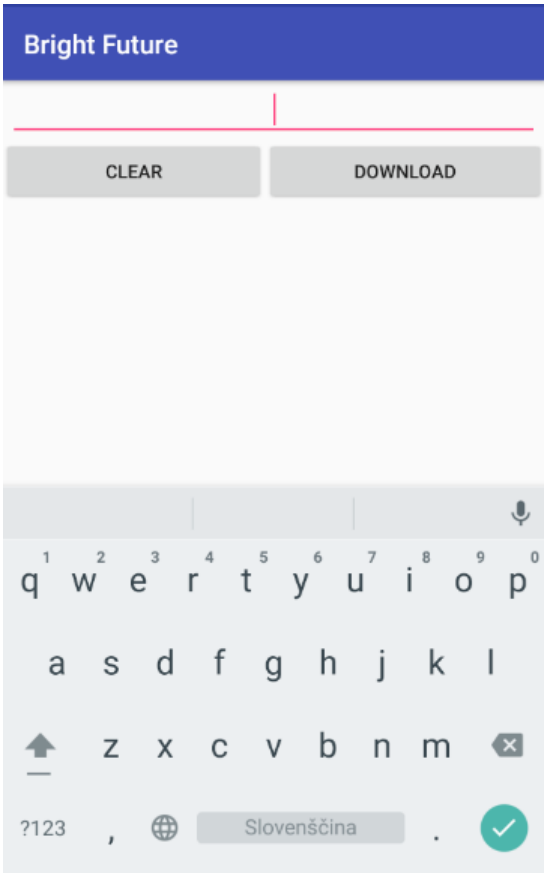
```
private void RunDBData(final int who) {
    final String[] DBTables = {"Sola", "Stroka", "Sola_Smer"};
    if (who < 3) {
        Runnable DBRun = () → {
                new TextExtractor(DBTables[who], getBaseContext(), PutIntoSQL, who);
        };

        Thread DBThread = new Thread(DBRun);
        DBThread.start();
    }
```

S26: Metoda, ki kliče metodo odgovorno za pridobivanje podatkov

S končanjem ponavljanja zgornje metode so v podatkovni bazi shranjeni vsi podatki o šolah in njihovih strokah. Aplikacija še nato ustvari novi poslušalec dogodkov za spremembe vnosa v iskanem polju aplikacije.

```
    TableSelectionText.addTextChangedListener(new TextWatcher() {
        @Override
        public void beforeTextChanged(CharSequence charSequence, int i, int i1, int i2) {

        }

        @Override
        public void onTextChanged(CharSequence charSequence, int i, int i1, int i2) {
            DisplayLayout.removeAllViewsInLayout();
            if (charSequence.length() > 1) {

                createArrayOfDataforDisplay(charSequence.toString());

            }
        }

        @Override
        public void afterTextChanged(Editable editable) {

        }
    });

}
```

S27: Poslušalec odgovoren za posodabljanje prikaz podatkov ob spremembi iskanega niza



S28: Izgled aplikacije in iskanega polja

V primeru, da je v iskanem polju več kot en znak se bo zagnala metoda createArrayOfDataforDisplay, ki jemlje kot parameter vnos v iskanem polju. Metoda createArrayOfDataforDisplay ustvari dve spremenljivki, ki bosta držali podatke, ki so pridobljeni od poizvedbe nad imeni strok in imeni šol v lokalni podatkovni bazi na mobilni napravi.

```
private void createArrayOfDataforDisplay(String PassedStringSearch) {
    //Log.i("LOG", "Search now reading: " + PassedStringSearch);
    String QueryCondtion = "LIKE \"%" + PassedStringSearch + "%\"";

    String SolaImeQuery = "Ime " + QueryCondtion + " OR Kratka_Oznaka " + QueryCondtion;
    String StrokaQuery = "Ime_Stroke " + QueryCondtion;

    Cursor EntriesSolaIme = DBEntryHandler.customQuery( table: "Sola",  query: "Ime", SolaImeQuery);
    Cursor EntriesStroka = DBEntryHandler.customQuery( table: "Stroka",  query: "Ime_Stroke", StrokaQuery);

    createTextViews(EntriesSolaIme, EntriesStroka);


}
```

S29: Metoda odgovorna za iskanje podakotv in njihovo prikazovanje glede na iskan niz, ki ga je vnesel uporabnik

Te podatke nato pošlje naprej v metodo createTextViews, ki za obojne podatke ustvari opazljiv in interaktivni list vseh rezultatov iskanja.

```
private void createTextViews(Cursor EntriesSola, Cursor EntriesStroka) {

    if (EntriesSola.moveToFirst()) {      //sola
        do {
            TextView DisplayValue = new TextView( context: this);
            final String EntryValue = EntriesSola.getString( i: 0);
            String displayText = EntryValue + " - Sola";
            DisplayValue.setText(displayText);
            DisplayValue.setBackgroundResource(R.color.grayinsh);
            DisplayValue.setLayoutParams(GlobalTextDisplayParam);
            DisplayValue.setPadding( left: 20,  top: 15,  right: 20,  bottom: 15);


            DisplayValue.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View view) {
                    Intent i = new Intent(getBaseContext(), SchoolDetails.class);
                    i.putExtra( name: "ImeSole", EntryValue);
                    startActivity(i);
                }
            });

            //Log.i("LOG", "Logging: " + EntriesSola.getColumnName(0) + " - " + EntriesSola.getString(0));
            DisplayLayout.addView(DisplayValue);
        } while (EntriesSola.moveToNext());

    }

    if (EntriesStroka.moveToFirst()) {
        do {
            TextView DisplayValue = new TextView( context: this);
            final String EntryValue = EntriesStroka.getString( i: 0);
            String displayText = EntryValue + " - Stroka";
            DisplayValue.setText(displayText);
            DisplayValue.setBackgroundResource(R.color.grayinsh);
            DisplayValue.setLayoutParams(GlobalTextDisplayParam);
            DisplayValue.setPadding( left: 20,  top: 15,  right: 20,  bottom: 15);

            DisplayValue.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View view) {
                    Intent i = new Intent(getBaseContext(), StrokaDetails.class);
                    i.putExtra( name: "ImeStroke", EntryValue);
                    startActivity(i);
                }
            });

            //Log.i("LOG", "Logging: " + EntriesStroka.getColumnName(0) + " - " + EntriesStroka.getString(0));
            DisplayLayout.addView(DisplayValue);
        } while (EntriesStroka.moveToNext());
    }
```

S30: Metodi odgovorni za ustvarjanje vidnih rezultatov

S31: Vidni rezultati v aplikaciji glede na iskan niz

Ob kliku na rezultat, ki je bil pridobljen iz imena sole, se ta odpre v novi zaslon, ki ima vse podatke o šoli, kateri je bila izbrana. Prikazane so tudi vse stroke, ki so na voljo za obiskovanje na tej šoli in hkrati tudi število vpisnih mest.

S32: Izgled aplikacije ob kliku na rezultat šole

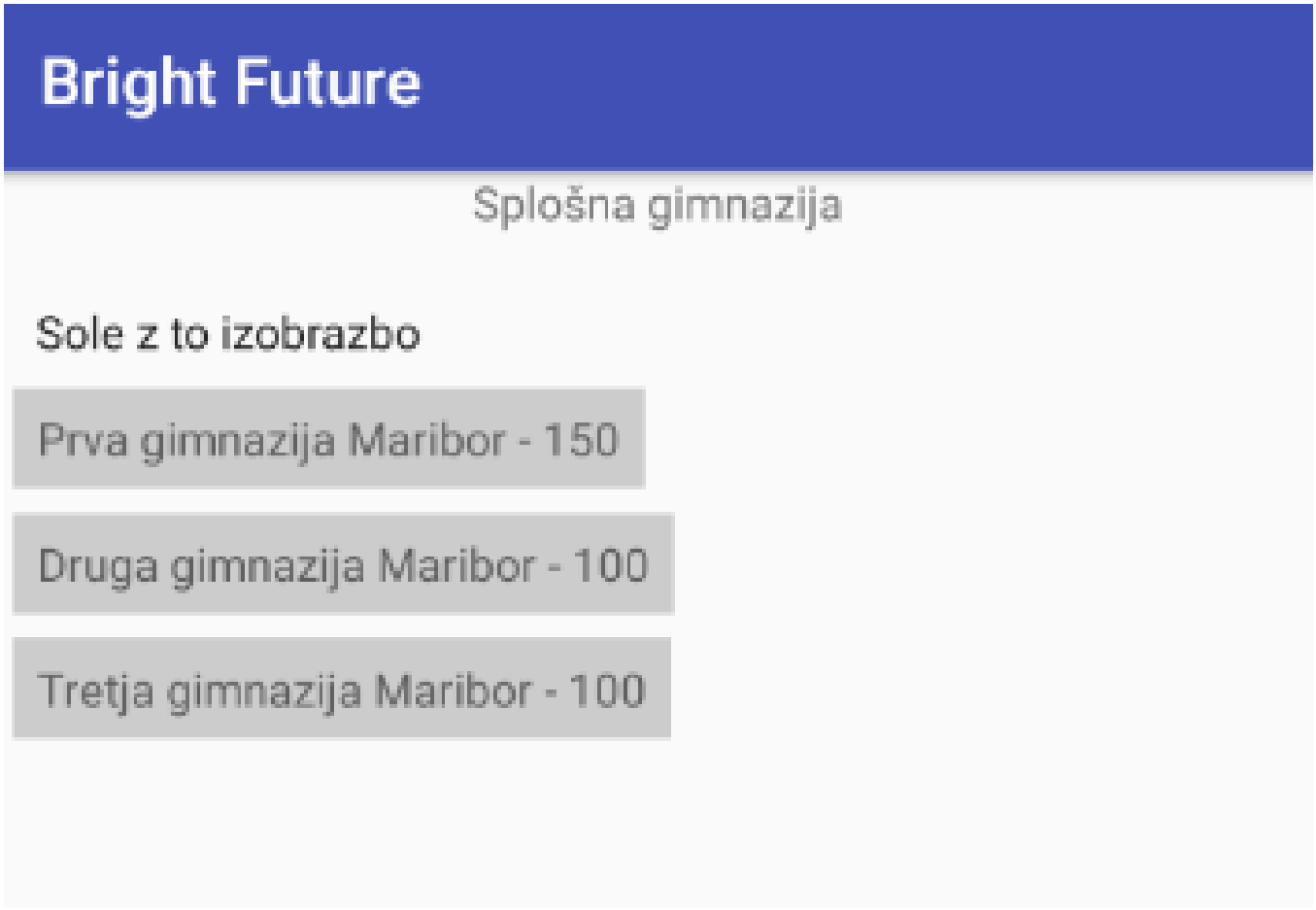
```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_school_details);

    SchoolName = findViewById(R.id.SchoolName);
    StrokaLayout = findViewById(R.id.StrokeLayout);


    Intent intent = getIntent();

    String Sola = intent.getStringExtra( name: "ImeSole");

    SchoolName.setText(Sola);

    displayResultsForSola(Sola);


}


private void displayResultsForSola(String sola){
    Cursor Entries = MainActivity.DBEntryHandler.getStrokaFromSchool(sola);

    if(Entries.moveToFirst()){
        do{
            final String imeStroke = Entries.getString(Entries.getColumnIndex( s: "Ime_Stroke"));
            final int SteviloMest = Entries.getInt(Entries.getColumnIndex( s: "Stevilo_mest"));
            TextView displayStroka = new TextView( context: this);

            displayStroka.setOnClickListener((view) → {
                    Intent i = new Intent(getBaseContext(), StrokaDetails.class);
                    i.putExtra( name: "ImeStroke", imeStroke);
                    startActivity(i);
            });
            String DisplayText = imeStroke + " - " + SteviloMest;
            displayStroka.setText(DisplayText);
            displayStroka.setBackgroundResource(R.color.grayinsh);
            displayStroka.setLayoutParams(MainActivity.GlobalTextDisplayParam);
            displayStroka.setPadding( left: 20,  top: 15,  right: 20,  bottom: 15);

            StrokaLayout.addView(displayStroka);
        }while (Entries.moveToNext());
    }
}
```

S33: Metodi odgovorni za prikaz novih podatkov glede na šolo, ki je bila podana ob kliku

Enako velja tuda za podatke pridobljene iz imen strok, vendar z razliko, da ob pritisku na ta tip rezultatov, se ti odprejo v novi zaslon z podatki o stroki in listom vseh šol s to stroko ter število razpoložljivih mest za to stroko.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_stroka_details);

    StrokaIme = findViewById(R.id.StrokaIme);
    StrokaLayout = findViewById(R.id.StrokeLayout);

    Intent intent = getIntent();
    String ImeStroke = intent.getStringExtra( name: "ImeStroke");

    StrokaIme.setText(ImeStroke);

    displayResultsForSola(ImeStroke);
}

private void displayResultsForSola(String stroke){
    Cursor Entries = MainActivity.DBEntryHandler.getSchoolFromStroka(stroke);

    if(Entries.moveToFirst()){
        do{
            final String imeSole = Entries.getString(Entries.getColumnIndex( s: "Ime"));
            final int SteviloMest = Entries.getInt(Entries.getColumnIndex( s: "Stevilo_mest"));
            TextView displayStroka = new TextView( context: this);
            String DisplayText = imeSole + " - " + SteviloMest;
            displayStroka.setText(DisplayText);
            displayStroka.setBackgroundResource(R.color.grayinsh);
            displayStroka.setLayoutParams(MainActivity.GlobalTextDisplayParam);
            displayStroka.setPadding( left: 20,  top: 15,  right: 20,  bottom: 15);

            displayStroka.setOnClickListener((view) → {
                    Intent i = new Intent(getBaseContext(), SchoolDetails.class);
                    i.putExtra( name: "ImeSole", imeSole);
                    startActivity(i);
            });

            StrokaLayout.addView(displayStroka);
        }while (Entries.moveToNext());
    }
}
```

S34: Metodi za prikaz novih podatkov glede na stroko, ki je bila podana ob kliku



S35: Prikaz rezultatov v aplikaciji

Obrazložitev pridobitve podatkov iz HTML kode

https://illegible-deflector.000webhostapp.com/Site3/dbtest.php?Tabela=

Doda se Ime tabele k prejšnjmu nizu

Ime tabele, ki je iskana — Sola

https://illegible-deflector.000webhostapp.com/Site3/dbtest.php?Tabela=Sola

```
private void ParseDBFront(final String link, final Context context) {
    Ion.with(context).load(link).asString().setCallback((e, result) -> {
            ////Log.i("LOG", "Log: " + result);
            Entries = dataHandler(result);
            SendToDataBase();

            Message msg = Message.obtain();
            msg.obj = iAm;
            DBHandler.sendMessage(msg);
    });
}
```

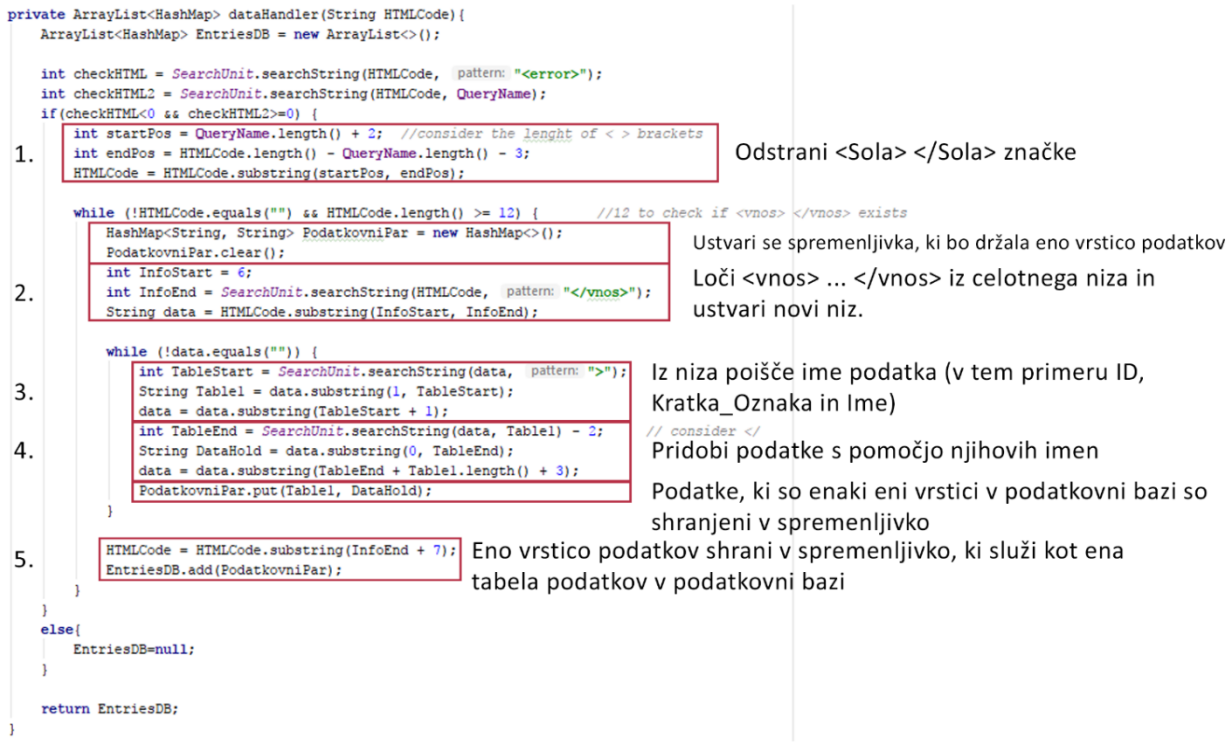
Asinhrona metoda vrne spremenljivko "result", ki HTML koda iz PHP strežnika

```
<Sola><vnos><ID>1</ID><Kratka_Oznaka>SERS</Kratka_Oznaka><Ime>Srednja Elektro Računalniška Šola</Ime></vnos>
```

Nato se kliče metoda "dataHandler", ki sprejme HTML niz, ki je bil pridobljen in nastavi vrednost spremenljivke "Entries" na podatke, ki so bili pridobljeni iz HTML niza

```
private ArrayList<HashMap> dataHandler(String HTMLCode){
    ArrayList<HashMap> EntriesDB = new ArrayList<>();

    int checkHTML = SearchUnit.searchString(HTMLCode, pattern: "<error>");
    int checkHTML2 = SearchUnit.searchString(HTMLCode, QueryName);
    if(checkHTML<0 && checkHTML2>=0) {
        int startPos = QueryName.length() + 2;  //consider the lenght of < > brackets
        int endPos = HTMLCode.length() - QueryName.length() - 3;
        HTMLCode = HTMLCode.substring(startPos, endPos);

        while (!HTMLCode.equals("") && HTMLCode.length() >= 12) {       //12 to check if <vnos> </vnos> exists
            HashMap<String, String> PodatkovniPar = new HashMap<>();
            PodatkovniPar.clear();
            int InfoStart = 6;
            int InfoEnd = SearchUnit.searchString(HTMLCode, pattern: "</vnos>");
            String data = HTMLCode.substring(InfoStart, InfoEnd);

            while (!data.equals("")) {
                int TableStart = SearchUnit.searchString(data, pattern: ">");
                String Tablel = data.substring(1, TableStart);
                data = data.substring(TableStart + 1);
                int TableEnd = SearchUnit.searchString(data, Tablel) - 2;     // consider </
                String DataHold = data.substring(0, TableEnd);
                data = data.substring(TableEnd + Tablel.length() + 3);
                PodatkovniPar.put(Tablel, DataHold);
            }

            HTMLCode = HTMLCode.substring(InfoEnd + 7);
            EntriesDB.add(PodatkovniPar);
        }
    }
    else{
        EntriesDB=null;
    }

    return EntriesDB;
}
```

S36: Potek pridobitve podatkov 1



S37: Potek pridobitve podatkov 2

1. HTML Koda (Odstranjen del niza je označen *poševno*)

<Sola><vnos><ID>1</ID><Kratka_Oznaka>SERS</Kratka_Oznaka><Ime>Srednja Elektro Računalniška Šola</Ime></vnos><vnos><ID>2</ID><Kratka_Oznaka>PGM</Kratka_Oznaka><Ime>Prva gimnazija Maribor</Ime></vnos>***</Sola>***

2. Novi niz iz HTML kode

<vnos><ID>1</ID><Kratka_Oznaka>SERS</Kratka_Oznaka><Ime>Srednja Elektro Računalniška Šola</Ime></vnos>

2. Stari niz iz HTML kode ostane nespremenjen

<vnos><ID>1</ID><Kratka_Oznaka>SERS</Kratka_Oznaka><Ime>Srednja Elektro Računalniška Šola</Ime></vnos><vnos><ID>2</ID><Kratka_Oznaka>PGM</Kratka_Oznaka><Ime>Prva gimnazija Maribor</Ime></vnos>

3. Ime podatkov je poiskano iz novega niza s pomočjo HTML značk (en podatek na enkrat)

<**ID**>1</ID>

4. S pomočjo imena pridobljenega iz prejšnje točke pride program do podatka 1

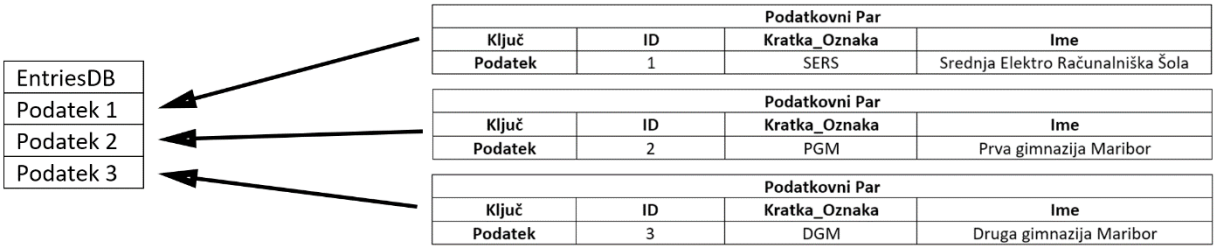
Podatke tudi shrani s svojim ključem v spremenljivko »Podatkovni Par« tipa HashMap

Podatkovni Par			
Ključ	**ID**	**Kratka_Oznaka**	**Ime**
Podatek	1	SERS	Srednja Elektro Računalniška Šola

5. Iz stare HTML kode odstrani vnos, ki je bil predelan in podatke shrani v spremenljivko »EntriesDB«

<vnos><ID>2</ID><Kratka_Oznaka>PGM</Kratka_Oznaka><Ime>Prva gimnazija Maribor</Ime></vnos>

Tako si je možno grafično predstavljati kako so podatki shranjeni v novo spremenljivko



S38: Grafični prikaz načina shranjevanja podatkov

Podatki se nehajo zapisovat v novo spremenljivko in so vrnjeni takrat, ko so izpolnjeni naslednji pogoji:

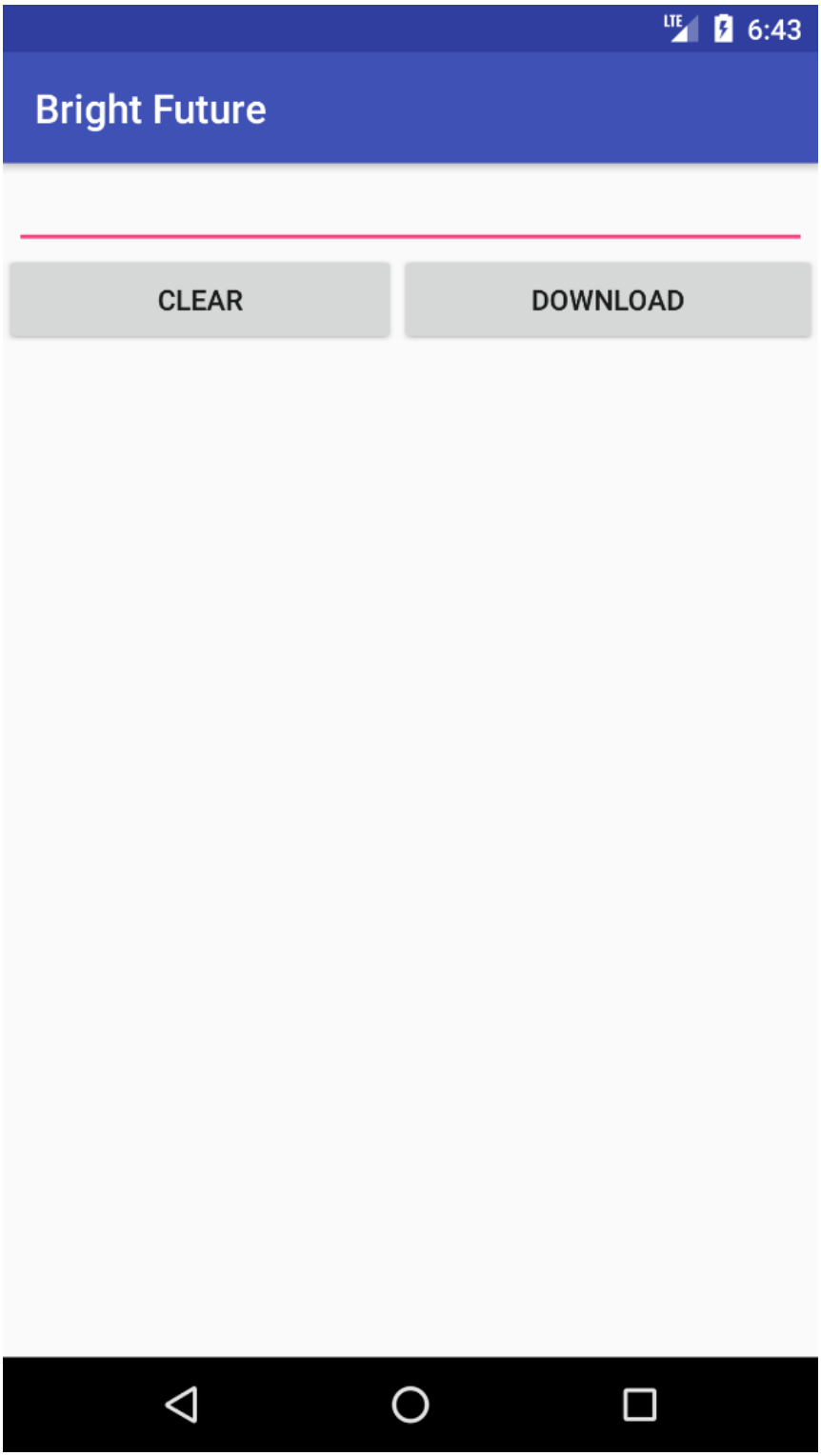
```
while (!HTMLCode.equals("") && HTMLCode.length() >= 12) {
```

S39: Pogoj za obdelavo podatkov

Kadar je dolžina HTML niza krajša od 12 znakov ali kadar je HTML niz prazen.

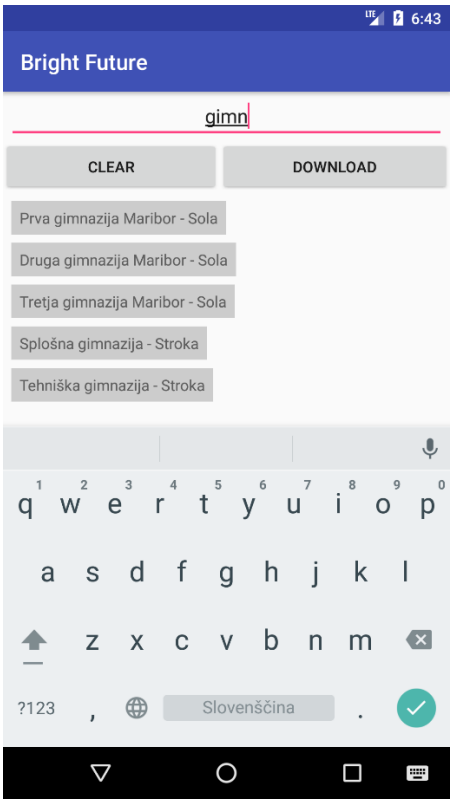
UPRAVLJANJE APLIKACIJE

V trenutni fazi je aplikacija še v zelo primitivni obliki in ne ponuja veliko možnosti uporabniku. V trenutnem stanju aplikacija omogoča najbolj osnovne možnosti iskanja šol in strok. Trenutno, ko se aplikacija zažene, bo ta posodobila svojo lokalno podatkovno bazo, kolikor je na voljo internetna povezava.



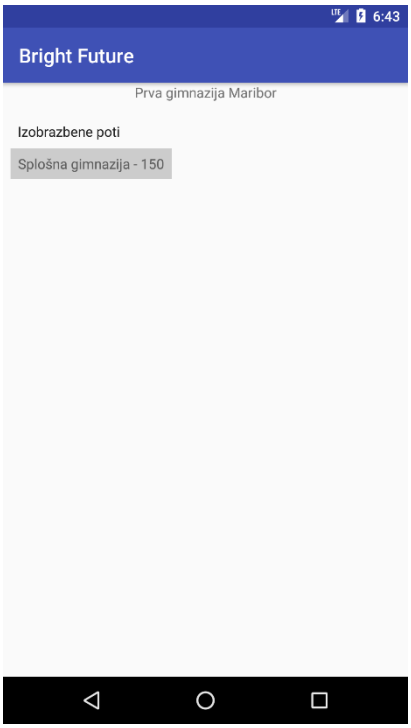
S40: Aplikacija ob zagonu

Ob vnašanju iskalnih parametrov, kot so kratica srednje šole ali del imena srednje šole, bo aplikacija vrnila vse rezultate, ki se ujemajo s iskanim nizom. Aplikacija bo že od vsega začetka iskala šole in stroke skupaj in jih skupaj tudi prikazovala.



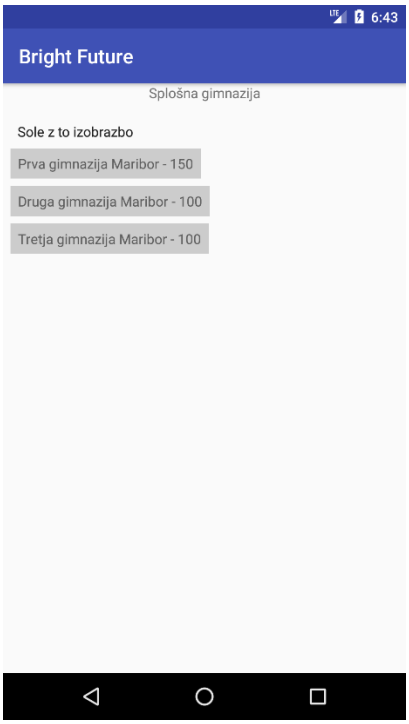
S41: Aplikacija ob iskanju šole ali stroke

Ob prikazu podatkov je možno na njih tudi pritisniti za dodatne informacije o teh strokah ali šolah.



S42: Aplikacija ob pritisku na rezultat tipa šole

V trenutni različici bo aplikacija prikazala samo ime srednje šole in njihove izobrazbene poti. Aplikacija tukaj omogoča klik na izobraževalne poti za lažjo navigacijo. V primeru, da uporabnik pritisne na eno izmed izobraževalnih poti, bo aplikacija prikazala vse srednje šole z to izobraževalno potjo.



S43: Aplikacija ob pritisku na rezultat tipa stroke

Aplikacija še omogoča, da se pritisne na novo prikazane šole in prikaz informacij o teh šolah ob kliku na njih.

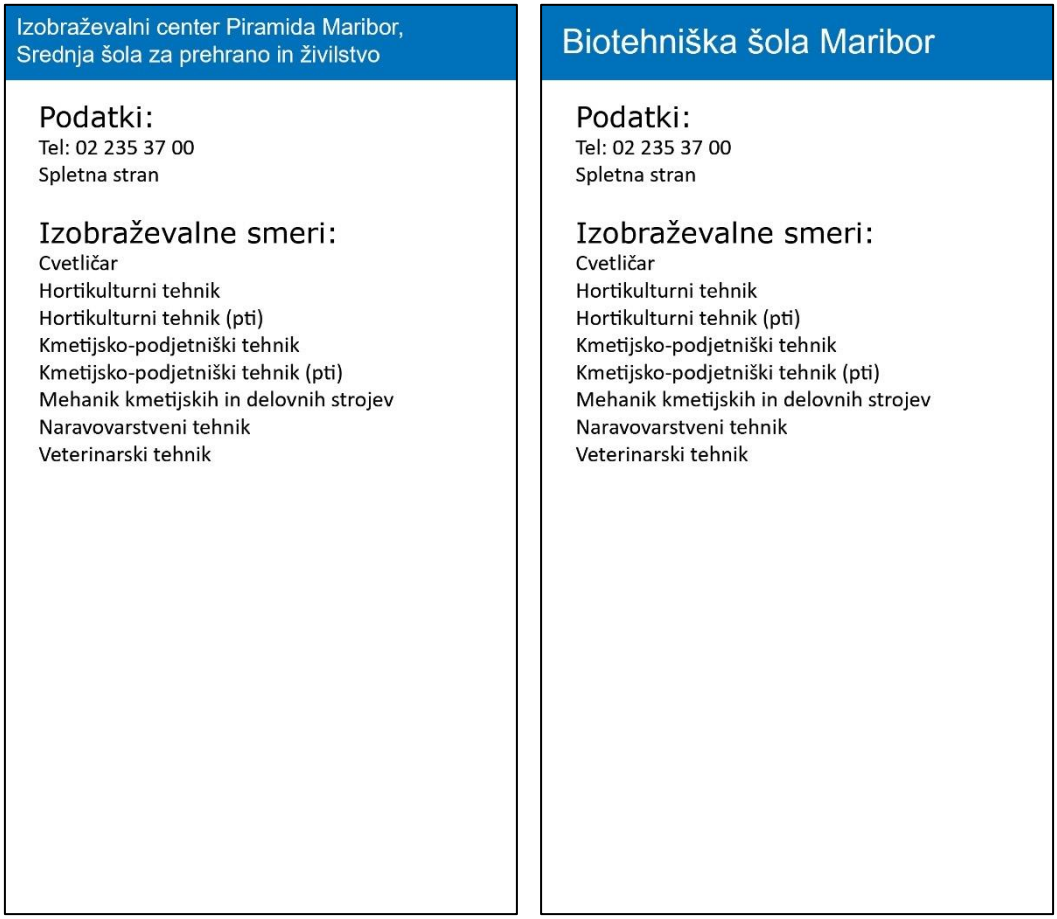
S44: Aplikacija ob ponovnem pritisku na rezultat

NADALJNI NAČRT RAZVIJANJA APLIKACIJE

Uporabniški vmesnik

Trenutna verzija aplikacije ima zelo grob uporabniški vmesnik, saj se navadni uporabnik z njim ne bi mogel znajti in bi se z njim mučil pri iskanju podatkov, ki jih išče. Zaradi tega je eden izmed prihodnjih ciljev ustvarjanje in načrtovanje uporabniškega vmesnika, ki bo omogočal lahko in hitro iskanje podatkov. Najboljši rezultat tega načrtovanja bi bil uporabniški vmesnik, ki bi na izgled lepo izpadel in se razlikoval od ostalih aplikacij, hkrati pa še vedno bil enostaven za uporabljat. Trenutne poskusne različice, ki prihajajo na pomislek pri načrtovanju uporabniškega vmesnika, se že razvijajo, vendar so še vedno premalo razvite in ne dosegajo učinka, ki je bil naveden.

Ena izmed različic, ki je prišla na razmislek pred kratkim je naslednja:



S45: Trenutno načrtovan uporabniški vmesnik

Vendar ta različica ima problem pri prikazovanju imen šole. Ena izmed situacij je prikazana v zgornji levi sliki, kjer je ime šole zelo dolgo in ga je bilo potrebno ločiti v dva odstavka ter pri tem zmanjšati velikost pisave. Ta pristop ni zaželjen, saj v tem primeru ne bodo vsi rezultati prikazani enotno. V vprašaj še prihaja tudi barvna tema in postavitev podatkov. Uporabniški vmesnik se bo razvijal sproti s aplikacijo dokler ne bo cilj uporabniškega vmesnika dosežen.

Optimizacija aplikacije

Trenutno ena izmed največjih skrbi za aplikacijo je neoptimiziranost aplikacije. Največja skrb je pridobivanje podatkov in njihovo shranjevanje. Trenutni način pridobivanja podatkov deluje za majhno število podatkov, vendar to se lahko spremeni pri velikem številu podatkov ali pa pri starejšem telefonskem modelu. Trenutno je že načrtovana optimizacija trenutne kode za klicanje metode za pridobitev podatkov. V mislih je tudi popolna sprememba strežnika in načina pridobivanja podatkov. Sprememba pridobivanja podatkov je mišljena tako, da aplikacija ne bi več pretvarjala HTML kode in iz nje vlekla podatke, vendar bi se aplikacija povezala do PHP strežnika, ta pa bi nazaj vrnil podatke v določenem redu.

Potreben bo tudi ponoven pregled v prkazovanje podatkov. Trenutni način prikazovanja podatkov ne deluje asinhrono, kar pomeni, da ob velikem število podatkov, katerih mora aplikacija preiskati, lahko pride do daljših časov neodzivanja aplikacije. Asinhroni način prikazovanja podatkov bi to odpravil, saj bi ob spremembi iskanega niza prekinil prejšno poizvedno nad podatki. To bi bil eden izmed prihodnih ciljev, ki bo tudi pomemben za uporabniški vmesnik, saj ta se ne sme nehati odzivat.

DRUŽBENA ODGOVORNOST

Družbena odgovornost je odgovornost organizacije ali osebe za vplive njenih odločitev in dejavnosti na družbo in okolje. Pri razvijanju aplikacije sem se vprašal, kako bi bil družbeno odgovoren pri razvijanju aplikacije. Družbena odgovornost aplikacije se začne s tem, da aplikacija ne zbira osebnih podatkov o uporabnikih. Pri tem je prišla na misel tudi odgovornost uporabnike obvestit o tem, če aplikacija sploh lahko dostopa do nekaterih funkcih njihovega osebnega telefona.

ZAKLJUČEK

V trenutnem stanju razvijanju aplikacije je še veliko za naredit. Potrebno je še tudi več testiranja prihodnih različic aplikacije za odkrivanje in odpravljanje napak, ki se lahko pojavijo v aplikaciji. Trenutno najpomembnejši problem aplikacije je optimizacija trenutne kode, katera v trenutnem stanju potrebuje boljšo preglednost in ponovni pregled logike za njo. Največ optimizacije potrebuje pridobivanje podatkov iz podatkovne baze. Naslednji problem za odpravit je uporabniški vmesnik, da bodo uporabniki z veseljem brskali po aplikaciji. Trenutno je uporabniški vmesnik v fazi načrtovanja in še ni bil implementiran v samo aplikacijo. Trenutno uporabniški vmesnik je bil namenjen testiranju in bo v prihodnosti zamenjan s uporabniškim vmesnikom, ki bo deloval na bolj ali manj enak princip vendar bo imel predstavljiv izgled.