Проект

на тема

„Разработка на Android приложение за водене на birdwatching записки“

Нов български университет
Факултет: Бакалавърски факултет
Департамент: Информатика
Програма: Информатика
Курсове: CSCB762 – Операционни системи за мобилни устройства; CSCB763 – Разработване на мобилни приложения
Преподавател: гл. ас. д-р Мартин Иванов

Владислав Жасминов Тачев, F59631
Специалност: Информатика
Учебна година: 2014/2015
Семестър: Пролетен семестър
Индивидуални протоколи: 9877 и 9878

Съдържание на проекта

Въведение

Мобилните технологии са най-бързо развиващата се технология в историята на технологичното развитие. Те изпреварват с темповете си развитието всички останали технологични браншове, за което спомага много и миниатюризацията на чиповете, като настоящата технология позволява производство на чипове около 10nm. В световен мащаб интернет потреблението през мобилни устройства през 2014г за първи път задмина потреблението през персонални компютри, а в някои държави, като Индия и Египет, потреблението през мобилни устройства се равнява грубо на 3 до 4 пъти потреблението през всички останали платформи. Допълнително навлизането на автоматизирането на дома (home automation) и Интернет на нещата (The Internet of things (IoT)) в ежедневието под формата на „умни“ ключалки, термостати и т.н. устройства е един от страничните ефекти от навлизането на мобилни устройства в злободневието, поради което за не само ежедневните, но и за строго специализирани задачи като ssh клиенти, FTP сървъри, RD-клиенти, Wolfram Alpha клиенти, речници, учебници и т.н. има мобилни приложения. Тъй като аз се занимавам с wildlife фотография и съм запален birdwatcher бе логично тематиката на приложението, което избирам да напиша да бъде свързано по някакъв начин с това мое хоби.

Проблем и задача

Приложението, кръстено Bird Sighting Tracker представлява списък с всички наблюдения на птици подредени по хронологичен ред. Приложението има много практически приложения както в любителското наблюдение на птици, така и при провеждането на природозащитна дейност от неправителствени и хуманитарни организации в огромни мащаби и големи цели, като проследяване на миграции, опазване на видове и хабитати, определяне на антропогенното влияние от дадена сфера над дивата природа и т.н. Някои от конкретните възможни употреби на такова приложение биха били:

- Провеждане на Среднозимно преброяване на Водолюбивите видове птици – преброяване на северни и полярни видове, мигриращи през зимата в по-южни пояси (включително България), което служи освен за проследяване на популациите, миграционните пътища на видовете, така и за косвени белези за дългосрочни климатични промени (изместване на въздушни течения, зони с постоянно високо или ниско атмосферно налягане и т.н.)
- Провеждане на МОВП (Мониторинг на Обикновените Видове Птици) – ежегодна практика на Дружеството за защита на птици (БДЗП) за проследяване на популационните нива на обикновените видове птици, като по този начин се следи и прогнозират бедствия и нашествия от гризачи, насекоми и други вредители, свръхпопулация на дадени видове, което води до дизбалансиране на популационните равнища и

оттам до потенциални проблеми с реколтата на житни култури например.
- Провеждането на любителски и професионален мониторинг на различни видове (не само птици) с цел откриване на постоянни хабитати за разработване на био, еко и фото-туризъм. Това приложение предполага и потенциално доста висока монетарна възвръщаемост на приложението.

Услуги обезпечени от продукта

Продуктът на този проект трябва да обезпечи основните услуги необходими за осъществяване на описване на наблюдения на птици (и други представители на фауната), като това не следва да компрометира удобството за потребителя. Тези следват да бъдат обобщени най-общо в:

- Създаване на запис
- Възможност за описване на
  - наблюдаван вид
  - брой представители на вида
  - място на наблюдението
  - време на наблюдението
- Възможност за изтриване на запис, без това да е прекалено лесно, за да се избегне потенциално изтриване по погрешка
- Възможност за сериализиране и запаметяване на данните, за да може да се възстанови предварително състояние на приложението
- Възможност за възпроизвеждането на данните от доставчик на съдържание (content provider)

Дизайн и архитектура на приложението

Тъй като приложението е мобилно, за съхранението на данните могат да се изберат различни платформи и методи: запомняне в локална база данни, описване на собствен клас сериализатор и десериализатор на данни, съхранение на отдалечен сървър за бази данни и дори клауд-базирани решения. Понеже приложението е относително малко и не предполага особено голямо натоварване, предвид бързодействието и относително малкия размер, избраният в случая подход е да се използва вградената в Android SQLite3 система за контрол на бази данни.

При реализацията на подобен тип приложения могат да се подемат различни похвати за запазване на данните, като много лесен е сериализирането на обектите от класа, който описва записите и записване на сериализираните обекти в базата данни, което е едно сравнително бързо и лесно решение на проблема, но относително ограничаващо бъдещото евентуално ъпгрейдване на модела на БД. Затова тук е използван малко по-сложния за реализация, но

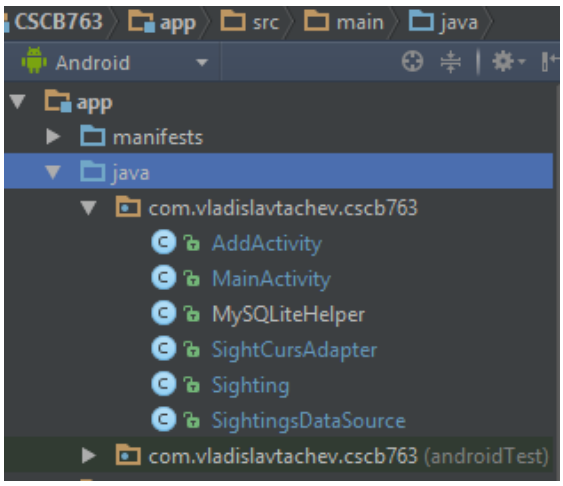
много по-гъвкав способ, в който всяко поле е отделна колона в таблицата. По този начин в последствие може да се добавят допълнителни таблици за различните потребители, както и да се добавят допълнителни услуги като например сортиране по други критерии, ъплоуд на записите през web API на сайт за онлайн тракинг на птици и т.н.

Практическа реализация

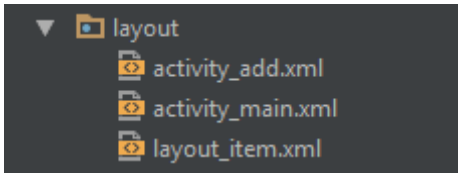
За практическата реализация на приложението е използвана една от най-известните и избрана за официална развойна среда за Android приложения – базираната на IntelliJ на JetBrains - Android Studio. За емулация, тестинг и дебъг на приложението първоначално бе използван ускорен чрез Intel HAXM емулатор, който е 4 пъти по-бърз от стандартната Андроид виртуална машина, но в последствие работата бе продължена върху Oracle VirtualBox базирания продукт Genymotion, който дава два до три пъти по-висока производителност и бързореакция при стартиране и инсталация на приложения, а в крайния си етап, тестването бе усъществено върху физически устройства: Google Nexus 4, работещ с vanilla Android Lollipop 5.1.1 и HTC One m7 работещ с 5.1.1 базиран CyanogenMod 12.1, за да е сигурно, че приложението ще върви без проблеми на всякакви устройства.

Проектът таргетира последните версии на развойния пакет, тъй като много използва т.нар. Android Material UI, което е концепция на сливане на основния потребителски интерфейс на устройството (бутони, лента за нотификации и т.н.) цветово и стилово с остатъка на приложението, но минималната версия на SDK заложена в проекта е 8, тъй като това е добрата практика за максимална съвместимост с Android устройствата ползващи Google Play понастоящем.

Проектът се състои от сумарно 6 класа и 3 лейаут XML елемента:



Фиг. 1 - Java-специфичната структура на проекта



Фиг. 2 - Трите XML Layout елемента на приложението

MainActivity и activity_main.xml

MainActivity.java е мейн активити класа за приложението и е активитито, което се зарежда по подразбиране при стартиране на приложението. Тъй като активито се състои от само няколко елемента, основният от които е ListView-то, което помещава списъка с наблюдения, класът разширява ListActivity класа.

Класа описва само необходимите методи на класа родител, както и няколко допълнителни, необходими за функционирането:

Методи на класа

onCreate() е основния метод на класа, като указва какво да се случи при създаването на инстанция на класа. Подава се стандартния Bundle параметър, който се грижи за правилно възстановяване на данните в случай на унищожаване на обекта докато е в паузирано състояние. Освен това се декларира DataSource, Cursor и CursorAdapter обекти:

- DataSource се грижи за правилната интеракция с базата данни и намаляване на грешките, затваряйки операциите в детерминитивна система с краен брой операции.
- Cursor се грижи за обхождане на резултата от SQLite заявката, като по този начин от една страна намалява предимно необходимото време за обработка на заявката, връщайки първия ред и продължавайки обработката в бекграунда. Това е добра практика с углед на това, че по този начин не се изчаква извършването на цялата заявка, а същевременно се избягва възможността за асинхронен достъп до базата данни от друга нишка или процес, което би могло да доведе до неконсистентност на данните, тъй като заявката е подадена за цялата база/таблица.
- CursorAdapter се грижи за визуализирането на данните от резултата и свързването им към собствен custom layout – layout_item.xml, който е обсъден по-натам в подробност.

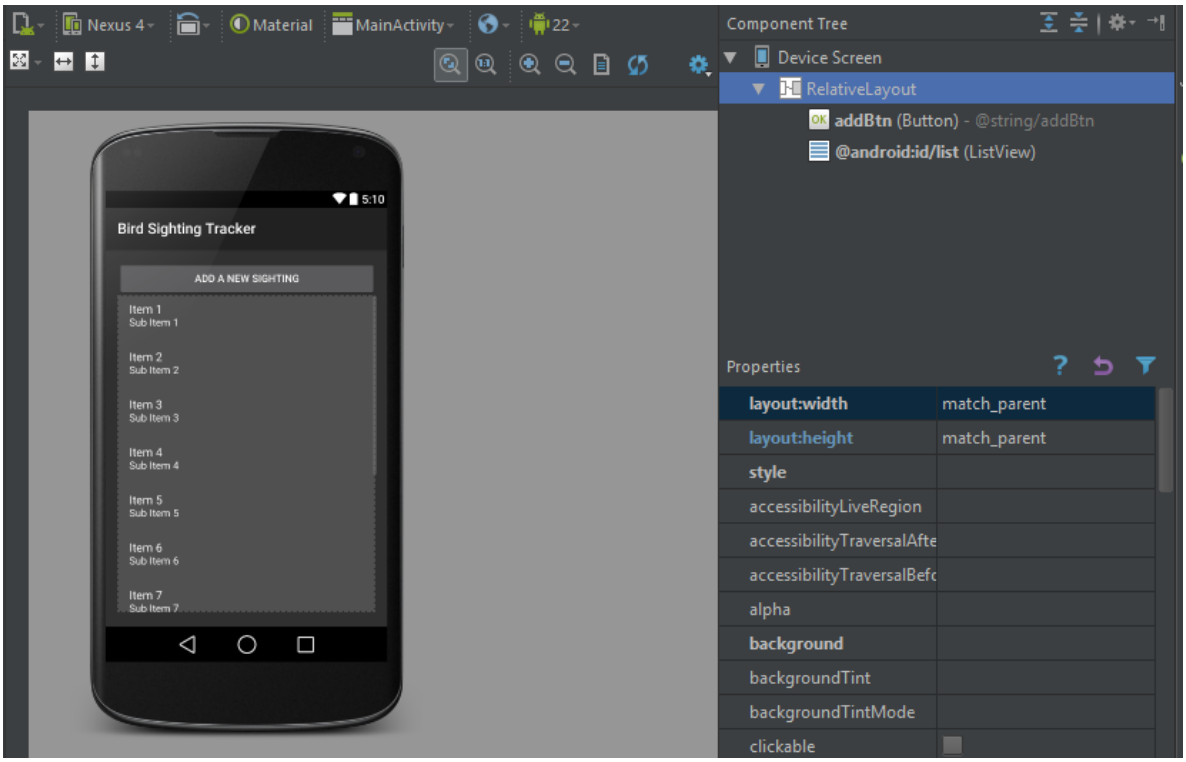
Също така методът помещава слушател за клик събития, който инициализира второ активити и прехвърля фокуса към него – AddActivity (разгледано след малко).

Допълнително класа има два метода – deletebuttonShow() и deleteEntry(), които се грижат съответно да слушат за longClick събитие и да покажат или скрият бутона за изтриване на съответния изглед/обект и да слушат за клик събития върху бутона, за да премахнат от базата данни съответния запис.

```
package com.vladislavtachev.cscb763;

import android.app.ListActivity;
import android.content.Intent;
import android.database.Cursor;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.ListView;
import java.util.ArrayList;
import java.util.List;

/**
 * Created by Vladislav Tachev on 6/29/2015.
 * All rights reserved.
 */

public class MainActivity extends ListActivity {

    public static final List<Sighting> Sightings = new ArrayList<Sighting>();
    private SightCursAdapter adapter = null;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        SightingsDataSource datasource = new SightingsDataSource(this);
        datasource.open();
        Cursor cursor = datasource.getAllSightingsAsCursor();
        ListView listView = (ListView) findViewById(android.R.id.list);
        listView.setLongClickable(true);
        adapter = new SightCursAdapter(this, cursor);
        listView.setAdapter(adapter);

        Button addBtn = (Button) findViewById(R.id.addBtn);
        addBtn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent addAct = new Intent(getApplicationContext(),
AddActivity.class);
                startActivity(addAct);
            }
        });
    }

    public void deletebuttonShow(View view) {
       final View delbutton =  view.findViewById(R.id.delBtn);
        view.setOnLongClickListener(new View.OnLongClickListener() {
            @Override
            public boolean onLongClick(View v) {
                if (delbutton.getVisibility()== View.INVISIBLE) {
                    delbutton.setVisibility(View.VISIBLE);
                } else delbutton.setVisibility(View.INVISIBLE);
                return true;
            }
        });
    }

    public void deleteEntry(View view) {
      adapter.deleteItem((Long) view.getTag(), this);
        SightingsDataSource datasource = new SightingsDataSource(this);
        datasource.open();
        Cursor cursor = datasource.getAllSightingsAsCursor();
        adapter.swapCursor(cursor);

    }
}
```

Както вече беше споменато layout-ът на активитито не се състои от много обекти:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="16dp"
    android:paddingBottom="16dp"
    android:orientation = "vertical"
    tools:context=".MainActivity" >

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/addBtn"
        android:id="@+id/addBtn"
        android:layout_alignParentTop="true"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_alignParentRight="true"
        android:layout_alignParentEnd="true" />

    <ListView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@android:id/list"
        android:layout_below="@+id/addBtn"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true" />

</RelativeLayout>
```



Фиг. 3 - Изчистен и семпъл изглед на основното активити, състоящо се от един бутон и ListView, което се популира динамично по време на създаването на инстанцията на активитито.

AddActivity и activity_add.xml

AddActivity.java е второто активити на приложението, което се състои само от два метода и реалната му цел е да прави връзката с datasource обекта, който описва интеракцията с базата данни и DBHelper-а, както и да подава заявка за записване на новото наблюдение.

Методи на класа

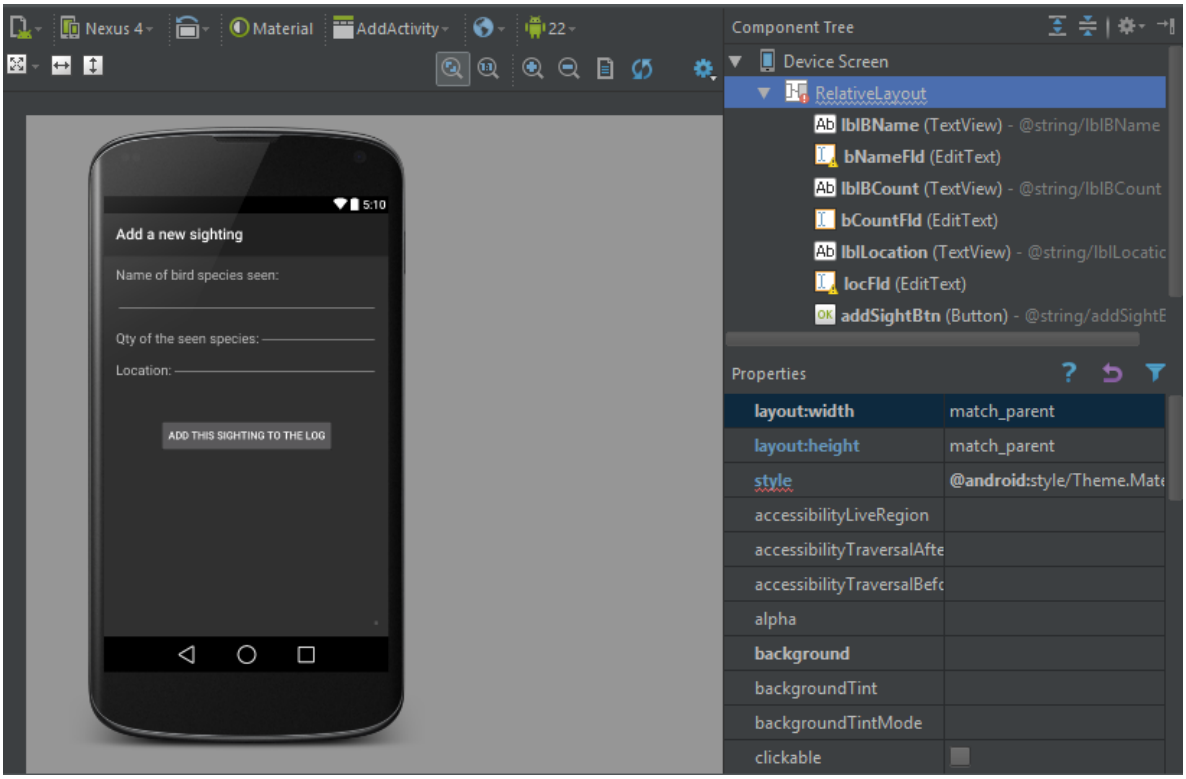
Като всяко активити и AddActivity трябва да оувъррайдне основния метод за класа родител – onCreate(). Тук също се подава обект Bundle за правилно възстановяване на данните, в случай че обектът е унищожен от garbage collector-а в състояние на пауза в стака на операционната система. Меътодът също се грижи за слушателя на събития на бутона за добавяне на наблюдения.

При настъпване на клик събития върху бутона, слушателя извиква втория метод на класа – addSight(), който създава нова инстанция на DataSource класа, и отваря връзка с базата данни за създаване на нов запис в нея. За да се улесни потребителя, за време на наблюдението се счита настоящото локално време на пторебителя, като се взима предвид часовата зона, в която устройството се намира (обсъдено по-подробно по-нататък). След това методът приключва и подава сигнал на garbage collector-а да унищожи обекта след приключване на процедурата, която в последните си редове извиква нова инстанция на MainActivity класа, опреснявайки пасивно състоянието на списъка.

```
package com.vladislavtachev.cscb763;

import android.app.ListActivity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;

/**
 * Created by Vladislav Tachev on 6/29/2015.
 * All rights reserved.
 */

public class AddActivity extends ListActivity {

    private SightingsDataSource datasource;

    private static EditText locFld = null;
    private static EditText bNameFld = null;
    private static EditText bCountFld = null;
    private static Button addSightBtn = null;


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity_add);
        addSightBtn = (Button) findViewById(R.id.addSightBtn);
        addSightBtn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                addSight(v);
```

```
            }
        });
    }

    public void addSight(View view){
        datasource = new SightingsDataSource(this);
        datasource.open();
        locFld = (EditText) findViewById(R.id.locFld);
        bNameFld = (EditText) findViewById(R.id.bNameFld);
        bCountFld = (EditText) findViewById(R.id.bCountFld);
                datasource.addSighting(locFld.getText().toString(),
                        System.currentTimeMillis() / 1000L,
                        bNameFld.getText().toString(),
                        Integer.parseInt(bCountFld.getText().toString()));
        datasource.close();
        finish();
        Intent addAct = new Intent(getApplicationContext(), MainActivity.class);
        startActivity(addAct);

    }
}
```

Layout-а на активитито се състои от няколко елемента:



Фиг. 4 - Изглед на Layout-а на AddActivity.java

Sighting

Sighting.java е кратък клас, описващ само променливите, които ще бъдат записвани и извиквани от базата данни с автоматично генерирани и необходими конструктори и гетери и сетери:

```java
package com.vladislavtachev.cscb763;

/**
 * Created by Vladislav Tachev on 6/29/2015.
 * All rights reserved.
 */

public class Sighting {
    private long id;
    private String location;
    private long time;
    private String bName;
    private int bCount;

    public Sighting(String loc, long time, String bName, int bCount){
        setLocation(loc);
        setTime(time);
        setbName(bName);
        setbCount(bCount);
    }

    public Sighting(){}

    public long getId() {
        return id;
    }

    public void setId(long id) {
        this.id = id;
    }

    public String getLocation() {
        return location;
    }

    public void setLocation(String location) {
        this.location = location;
    }

    public long getTime() {
        return time;
    }

    public void setTime(long time) {
        this.time = time;
    }

    public String getbName() {
        return bName;
    }

    public void setbName(String bName) {
        this.bName = bName;
    }

    public int getbCount() {
        return bCount;
    }

    public void setbCount(int bCount) {
        this.bCount = bCount;
    }

    @Override
    public String toString(){
        return bName;
    }
}
```

MySQLiteHelper

MySQLiteHelper.java е имплементация на стандартния за операции със SQLite3 бази данни клас SQLiteOpenHelper. Той декларира няколко константи, с които се осъществява връзаката с базата данни, както и в случай на нейната липса ясъздава. Втория метод на класа е необходимата декларация за упоменаване на поведение в случай на промяна на модела на базата данни:

```
package com.vladislavtachev.cscb763;

import android.database.sqlite.SQLiteOpenHelper;
import android.database.sqlite.SQLiteDatabase;
import android.content.Context;

/**
 * Created by Vladislav Tachev on 6/29/2015.
 * All rights reserved.
 */

public class MySQLiteHelper extends SQLiteOpenHelper{

    public final static String DB_NAME = "sightings.db";
    public final static int DB_VER = 1;
    public final static String TABLE_SIGHTINGS = "sightings";
    public final static String COL_ID = "_id";
    public final static String COL_LOCATION = "_loc";
    public final static String COL_TIME = "_time";
    public final static String COL_BNAME = "_bname";
    public final static String COL_BCOUNT = "_bcount";

    private final static String DB_CREATE = "create table if not exists "
            + TABLE_SIGHTINGS + "("
            + COL_ID + " integer primary key autoincrement, "
            + COL_LOCATION + " text not null, "
            + COL_TIME + " integer not null, "
            + COL_BNAME + " text not null, "
            + COL_BCOUNT + " integer not null);";

    public MySQLiteHelper(Context context){
        super(context, DB_NAME, null, DB_VER);
    }

    @Override
    public void onCreate(SQLiteDatabase database){
        database.execSQL(DB_CREATE);
    }

    @Override
    public void onUpgrade(SQLiteDatabase database, int oldVer, int newVer){
        database.execSQL("DROP TABLE IF EXISTS " + TABLE_SIGHTINGS);
        onCreate(database);
    }
}
```

SightingDataSource

SightingDataSource.java е имплементация за стандарта за източника на данни. Той на практика е арбитрарен клас, който декларира и позволява само

краен/детерминистичен брой операции с базата данни намалявайки възможните грешки в последствие на неправилна употреба.

Методи на класа

Класът декларира няколко стандартни метода: конструктор, open(), close(), cursrorToSighting(), getAllSightingsAsCursor(), getAllSightingsAsList(), addSighting() и deleteEntry(). Контрукторът инстанциира нов DBHelper, който използва за интеракция с базата, като методите open() и close() отварят и затварят връзката с БД.

addSighting() получава 4 параметра по сигнатура: низ, който указва мястото на наблюдението (loc -> location), long указващ времето, низ за името на вида и целочислена стойност за броя представители на вида. По принцип тъй като се ползват unix timestamp-ове за запазване на времето, типа на променливата за времето може да се оптимизира и до целочислена стойност, но понеже андроид интерпретира времето в милисекунди, а не в секунди, както се генерира unix timestamp-а, това би предположило допълнителни структурни промени. Също така предвид факта, че приложението е малко като мащаби и размери, оптимизацията не би довела до съществено подобрение на бързореакцията на приложението.

cursorToSighting() връща и подрежда стойностите на колоните от един ред на резултата – курсора в обект от класа Sighting, като по този начин може да се интерпретира и прилага в контекста на приложението.

getAllSightingsAsList() връща като резултат списък (ArrayList) от типа Sighting, като подрежда резултатите от заявката към базата данни посредством cursorToSighting() метода.

getAllSightingsAsCursor() връща курсор при изпълнение на заявка за всички редове от таблицата на БД – на практика много сходен метод с горния, с изключението че връща друг тип резултат.

deleteEntry() е методът, който изпълнява заявката за премахване на ред от таблицата на базата данни. Приема като аргумент дълга целочислена стойност, която е поредния аутоинкрементален идентификатор на реда, който искаме да бъде премахнат.

```
package com.vladislavtachev.cscb763;

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.util.Log;
import java.util.ArrayList;
import java.util.List;

/**
 * Created by Vladislav Tachev on 6/29/2015.
 * All rights reserved.
 */
```

```
public class SightingsDataSource {

    //DB info
    private SQLiteDatabase database;
    private SQLiteOpenHelper dbHelper;
    private String[] allColumns = { MySQLiteHelper.COL_ID,
MySQLiteHelper.COL_LOCATION,
            MySQLiteHelper.COL_TIME, MySQLiteHelper.COL_BNAME,
MySQLiteHelper.COL_BCOUNT };

    public SightingsDataSource(Context context){
        dbHelper = new MySQLiteHelper(context);
    }

    public void open(){
        database = dbHelper.getWritableDatabase();
    }

    public void close(){
        dbHelper.close();
    }

    public void addSighting(String loc, long time, String bName, int bCount){
        ContentValues values = new ContentValues();
        values.put(MySQLiteHelper.COL_LOCATION, loc);
        values.put(MySQLiteHelper.COL_TIME, time);
        values.put(MySQLiteHelper.COL_BNAME, bName);
        values.put(MySQLiteHelper.COL_BCOUNT, bCount);
        long insertId = database.insert(MySQLiteHelper.TABLE_SIGHTINGS, null,
values);
        Cursor cursor = database.query(MySQLiteHelper.TABLE_SIGHTINGS,
allColumns, MySQLiteHelper.COL_ID
                + " = " + insertId, null, null, null, null);
        cursor.moveToFirst();
        Sighting newSighting = cursorToSighting(cursor);
        cursor.close();
    }

    private Sighting cursorToSighting(Cursor cursor){
        Sighting sighting = new Sighting();
        sighting.setId(cursor.getLong(0));
        sighting.setLocation(cursor.getString(1));
        sighting.setTime(cursor.getLong(2));
        sighting.setbName(cursor.getString(3));
        sighting.setbCount(cursor.getInt(4));
        return sighting;
    }

    public List<Sighting> getAllSightingsAsList(){
        List<Sighting> sightings = new ArrayList<Sighting>();
        Cursor cursor = database.query(MySQLiteHelper.TABLE_SIGHTINGS,
allColumns, null, null, null, null, null);
        cursor.moveToFirst();
        while (!cursor.isAfterLast()) {
            Sighting sighting = cursorToSighting(cursor);
            sightings.add(sighting);
            cursor.moveToNext();
        }
        cursor.close();
        return sightings;
    }

    public Cursor getAllSightingsAsCursor(){
        Cursor cursor = database.rawQuery("SELECT * FROM sightings", null);
        return cursor;
    }
```

```
    public void deleteEntry(long id){
        int resu = database.delete("sightings", "_id = ?", new
String[]{String.valueOf(id)});
        Log.v("Database", "Deleted rows: " + resu);
    }
}
```

SightCursAdapter

SightCursAdapter.java е курсор адаптер, който се използва за връзката между курсора, резултат от заявката към базата данни и ListView елементът, който представлява списъка от записи. Реално погледнато в случая курсор адаптерът изпълнява функцията на ListAdapter.

Методи на класа

Класът е осносително кратък и имплементира само трите необходими за класа родител метода – конструктор, newView() и bindView(), както и два допълнителни метода – deleteItem() и changeCursor()

Конструкторът създава обект от дефинирания подтип на CursorAdapter, като го вкарва в контекста на приложението.

newView() реално прави връзката между ListView родителския обект и XML layout-та, който трябва да бъде използван за запълване (inflate) на списъка.

bindView() свързва резултатите от курсора, който връща стойностите от заявката към базата данни с полетата на layout-а.

deleteItem() създава нов datasource обект, изтривайки ред от таблицата извиквайки метова deleteEntry() на SightingDataSource обекта. Допълнително се извиква кратко съобщение за нотификация тип Toast, посредство което се уведомява потребителя, че записът е премахнат сполучливо.

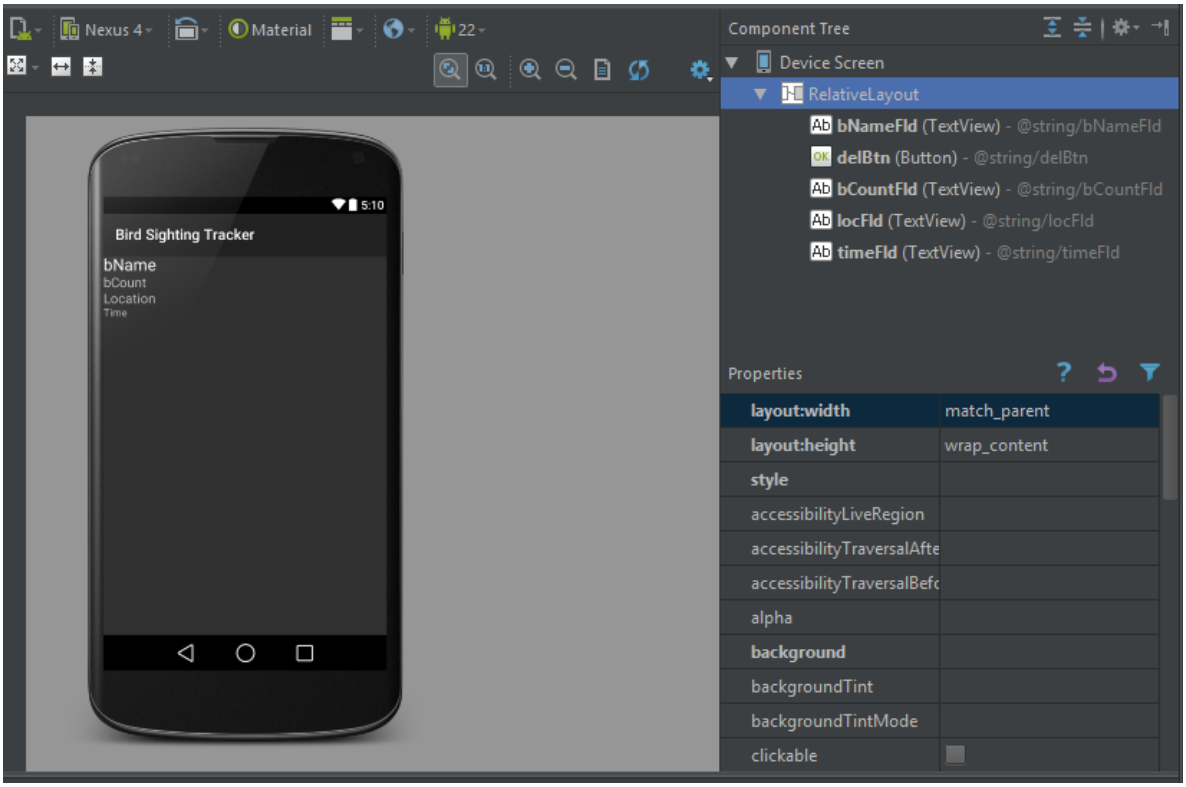
changeCursor() има за цел да опресни курсора, който се използва за запълване на ListView-то, като по този начин прави мигновено видима промяната от изтриване на ред от таблицата, без да се нуждае обекта на основното активити да бъде пресъздаван, което пести ресурси и процесорно време и би било видимо и усетено при по-слаби устройства.

```
package com.vladislavtachev.cscb763;

import android.content.Context;
import android.database.Cursor;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.CursorAdapter;
import android.widget.TextView;
import android.widget.Toast;

/**
 * Created by Vladislav Tachev on 7/2/2015.
 * All rights reserved.
 */
```

```
public class SightCursAdapter extends CursorAdapter {

    static Button delBtn;

    public SightCursAdapter(Context context, Cursor cursor) {
        super(context, cursor, 0);
    }

    @Override
    public View newView(Context context, Cursor cursor, ViewGroup parent){
        return LayoutInflater.from(context).inflate(R.layout.layout_item, parent,
false);
    }

    @Override
    public void bindView(View view, Context context, Cursor cursor){
        TextView tvLoc = (TextView) view.findViewById(R.id.locFld);
        TextView tvTime = (TextView) view.findViewById(R.id.timeFld);
        TextView tvBName = (TextView) view.findViewById(R.id.bNameFld);
        TextView tvBCount = (TextView) view.findViewById(R.id.bCountFld);
        delBtn = (Button) view.findViewById(R.id.delBtn);
        String loc = cursor.getString(cursor.getColumnIndexOrThrow("_loc"));
        long time = cursor.getInt(cursor.getColumnIndexOrThrow("_time"));
        String bName = cursor.getString(cursor.getColumnIndexOrThrow("_bname"));
        int bCount = cursor.getInt((cursor.getColumnIndexOrThrow("_bcount")));
        long id = cursor.getInt(cursor.getColumnIndexOrThrow("_id"));
        delBtn.setTag(id);
        tvLoc.setText(loc);
        tvTime.setText(String.valueOf(new java.util.Date(time*1000)));
        tvBName.setText(bName);
        tvBCount.setText(String.valueOf(bCount));
    }

    public void deleteItem(long itemID, Context context){
        SightingsDataSource dataSource = new SightingsDataSource(context);
        dataSource.open();
        dataSource.deleteEntry(itemID);
        Toast msg = Toast.makeText(context.getApplicationContext(),"Sighting
successfully deleted.", Toast.LENGTH_SHORT);
        msg.show();
    }

    @Override
    public void changeCursor(Cursor cursor) {
        super.changeCursor(cursor);
    }
}
```

layout_item.xml

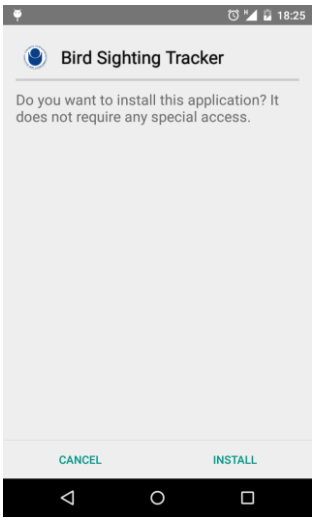
layout_item представлява специализиран частен изглед за изпълване на списъка на основното активити. По този начин се подреждат в контекст променливите на обекта Sighting, вкарвайки ги в ред по важност и специфичност за по-добра прегледност при разглеждане на записите. Така се постига наситеност на информация, без излишество на текст и пренатрупване на интерфейса, което и спомага потребителят да може да използва приложението без да се изгубва в излишен повтаряем текст, който не допринася по никакъв начин за употребата на приложението и четенето на записите.



Фиг. 5 - Изглед и реализация на елементите на списъка със специфична подредба и размери на елементите за по-добра прегледност

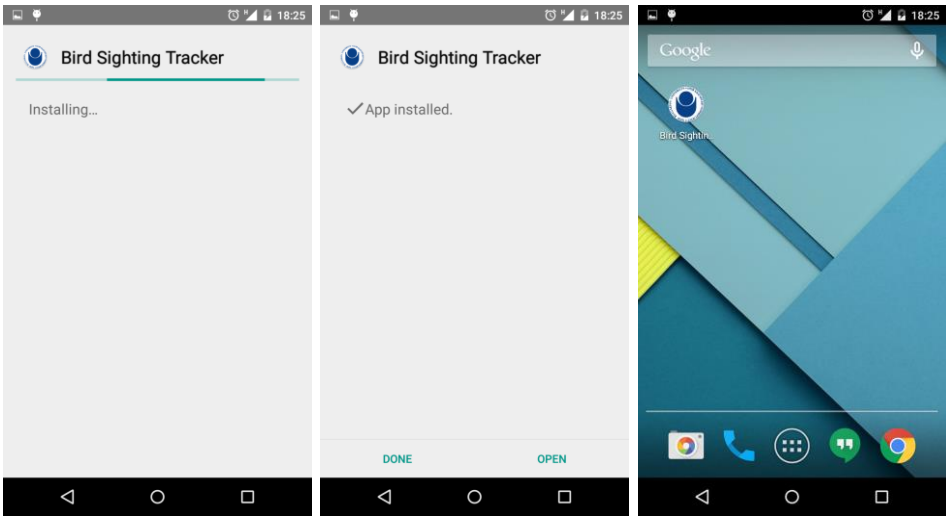
Примерна употреба на приложението

Тъй като след дебъг и тест на приложението е генериран дигитално подписан компилиран андроид пакет .apk, приложението е готово за качване в Google PlayStore или просто на сайт за сваляне по желание.



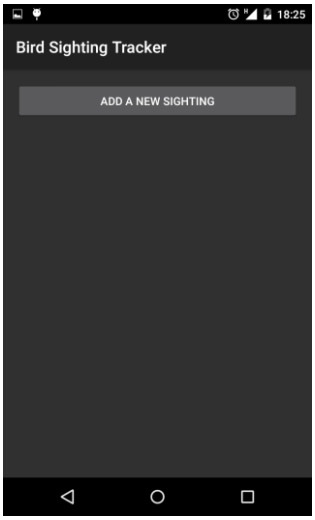
Фиг. 6 - Инсталационния екран на приложението върху физическо устройство от distirbutable APK

Тъй като приложението използва само частни за себе си услуги и не използва нативни услуги като камера, NFC четец, мрежа и т.н. AndroidManifest.xml не изисква допълнителни разрешения и съответно инсталационния екран не ги показва.



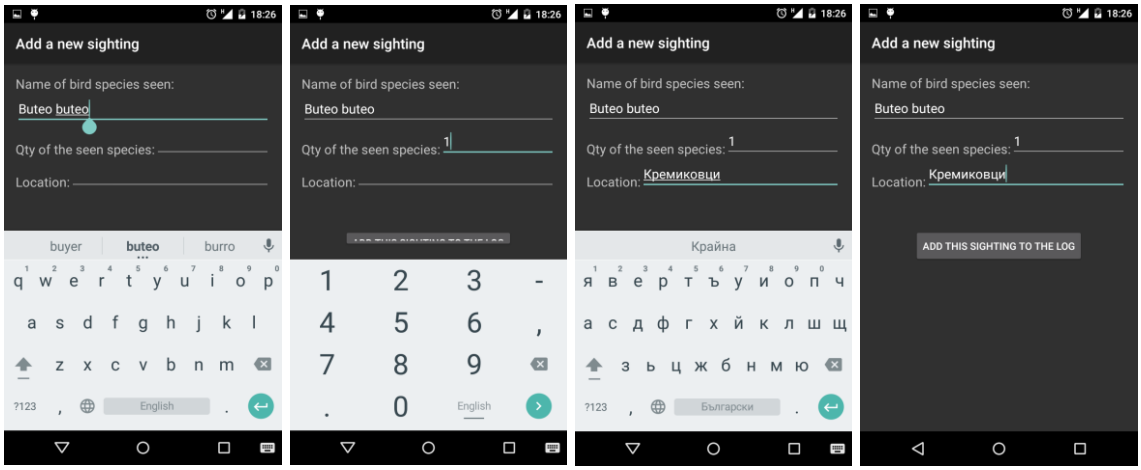
Фиг. 7 - Процес на инсталация на приложението и поставяне (ръчно) на икона върху работния плот

При стартиране на приложението, тъй като е първо стартиране, все още няма записи за показване и списъкът остава празен.



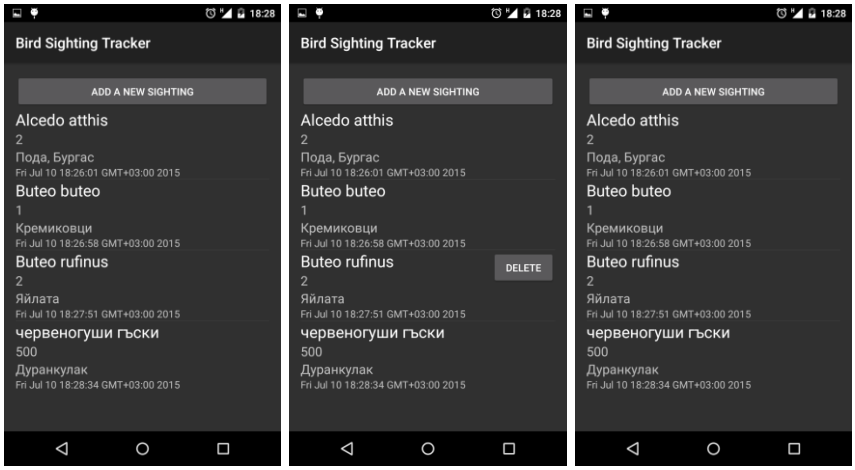
Фиг. 8 - Първи изглед на приложението с все още празен списък

При стартиране на добавянето на записи се зарежда изгледа на AddActivity



Фиг. 9 -Изглед на полетата за добавяне на запис към списъка.

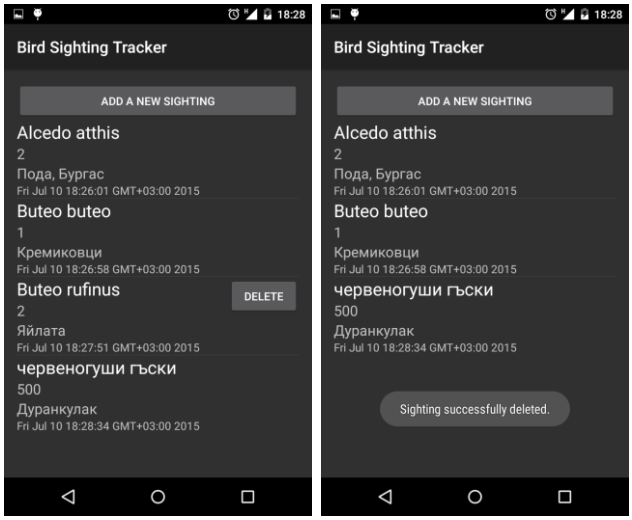
Както се вижда от снимките по-горе, тъй като ясно е декларирано какъв тип данни се очакват за всяко поле, системата позволява само цифрен запис броя наблюдавани индивиди, като по този начин се контролират входните данни към базата данни и се избягва необходимостта от допълнителни проверки на входните данни. По този начин няма потребителят да въведе грешни данни и да получи пост-фактум съобщение за грешка, или дори краш на приложението, правейки употребата на приложението много по-ясна, бърза и интуитивна за потребителя.



Фиг. 10 - Показване и скриване на бутона за изтриване при вече въведени в системата записи

След въвеждането на записи, списъкът придобива вида, който се очаква да има – на пълен с информация класьор, за провеждане на еко-научна дейност.

При продължително кликане върху някои произволен елемент от списъка, иначе скритият бутон за изтриване се появява, като по този начин се предотвратява нежелано изтриване на информация от базата данни. В случай на погрешно визуализиране на бутона, той може да бъде скрит отново посредством продължителен клик върху същия елемент.



Фиг. 11 - Изтриване на запис и Toast нотификацията, която последва изтриването

При натискане на бутона за изтриване, записът бива премахван и излиза кратковременно съобщение тип Toast, което уведомява потребителят, че съобщението е изтрито сполучливо.

Системен код и лицензи

Тъй като общия код на приложението възлиза на повече от 1000 реда, с цел спестяване на безцелен пълнеж, тук приложеният сорс код е сведен до абсолютния минимум. Останалия код е достъпен за преглед, инспекция и подобрения в github repository-то на проекта на адрес https://github.com/vlexster/Bird-Sighting-Log--NBU-CSCB763-Android-app-, където може да бъде намерен и подписан APK файл, готов за дистрибуция и инсталация на устройства.

Проектът не е реално лицензиран, въпреки че би спаднал към GNU GPL или GNU MIT лицензионните класове и поради публикуването си в github е практически отворен за колаборация, подобрения и персонализация.

Заключение

Проектът прилага основни концепции в проектирането на приложения за мобилни устройства с локално запазване на данни в нативната система за контрол на бази данни SQLite3. Приложението предоставя една стабилна и цялостна архитектура, с множество проверки и защити на данните за подсигуряване на тяхната хомогенност и предотвратяване на грешки при изпълнението на заявки. При изграждането му са спазени всички стандарти и добри практики за изграждане на мобилни приложения.