ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Департамент программной инженерии

СОГЛАСОВАНО
Доцент департамента программной инженерии факультета компьютерных наук, канд. техн. наук

_____ С. Л. Макаров
«___» _____ 202_ г.

УТВЕРЖДАЮ
Академический руководитель образовательной программы «Программная инженерия» профессор департамента программной инженерии, канд. техн. наук

_____ В.В. Шилов
«___» _____ 202_ г.

Подп. и дата	
Инв. № дубл.	
Взам. инв. №	
Подп. и дата	
Инв. № подл	

МОБИЛЬНОЕ ПРИЛОЖЕНИЕ, РАСПОЗНАЮЩЕЕ ГИПЕРССЫЛКИ ПО ФОТОГРАФИИ
Текст программы
ЛИСТ УТВЕРЖДЕНИЯ
RU.17701729.04.07-01 12 01-1-ЛУ

Исполнитель
студент группы _____
_____ / М. Я. Белкин /
«____»_____ 202_ г.

Москва 2019

УТВЕРЖДЕН
RU.17701729.04.07-01 81 01-1-ЛУ

МОБИЛЬНОЕ ПРИЛОЖЕНИЕ,
РАСПОЗНАЮЩЕЕ ГИПЕРССЫЛКИ ПО ФОТОГРАФИИ
Текст программы
RU.17701729.04.07-01 12 01-1
Листов 48

Подп. и дата	
Инв. № дубл.	
Взам. инв. №	
Подп. и дата	
Инв. № подл	

Москва 2020

СОДЕРЖАНИЕ

1. ТЕКСТ ПРОГРАММЫ

1.1. Analyzer.java

```
package com.belkin.linker;

import android.app.Activity;
import android.content.Context;
import android.graphics.Bitmap;
import android.media.Image;
import android.net.Uri;
import android.util.Log;
import android.widget.Toast;

import androidx.camera.core.ImageAnalysis;
import androidx.camera.core.ImageProxy;

import com.google.firebase.ml.vision.FirebaseVision;
import com.google.firebase.ml.vision.common.FirebaseVisionImage;
import com.google.firebase.ml.vision.common.FirebaseVisionImageMetadata;
import com.google.firebase.ml.vision.text.FirebaseVisionTextRecognizer;

import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.Objects;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

class Analyzer implements ImageAnalysis.Analyzer {

    private final static String CLASS_LOG_TAG = "Analyzer";
```

```
    /**
     * OCR ML model is downloaded to user device
     * Free
     */
    final int LOCAL_MODEL = 0;


    /**
     * OCR ML model is in cloud
     * Need payments
     *
     * @see "https://console.firebase.google.com/u/0/project/linker-
6ef34/overview"
     */
    final int CLOUD_MODEL = 1;


    private int model = LOCAL_MODEL;


    void useLocalModel() {
        Log.i(CLASS_LOG_TAG, "useLocalModel() method call");
        model = LOCAL_MODEL;
    }


    void useCloudModel() {
        Log.i(CLASS_LOG_TAG, "useCloudModel() method call");
        model = CLOUD_MODEL;
    }


    private Context context;
    private Activity activity;


    Analyzer(Activity activity) {
        super();
```

```
        this.activity = activity;
        this.context = activity.getBaseContext();
    }

    private int degreesToFirebaseRotation(int rotationDegrees) {
        Log.i(CLASS_LOG_TAG, "degreesToFirebaseRotation() method call");
        switch (rotationDegrees) {
            case 0:
                return FirebaseVisionImageMetadata.ROTATION_0;
            case 90:
                return FirebaseVisionImageMetadata.ROTATION_90;
            case 180:
                return FirebaseVisionImageMetadata.ROTATION_180;
            case 270:
                return FirebaseVisionImageMetadata.ROTATION_270;
            default:
                Log.e(CLASS_LOG_TAG, "Illegal argument! rotationDegrees = " +
rotationDegrees + " Rotation must be 0, 90, 180, or 270.");
                throw new IllegalArgumentException(
                        "Rotation must be 0, 90, 180, or 270.");
        }
    }

    @Override
    public void analyze(ImageProxy imageProxy, int degrees) {
        Log.i(CLASS_LOG_TAG, "analyze(ImageProxy, int) method call");

        if (imageProxy == null || imageProxy.getImage() == null) {
            return;
        }

        int rotation = degreesToFirebaseRotation(degrees);
```

```
        Image mediaImage = imageProxy.getImage();
        FirebaseVisionImage image =
FirebaseVisionImage.fromMediaImage(mediaImage, rotation);
        analyze(image);


    }


    public void analyze(String imagePath) {
        try {
            //FirebaseVisionImage image =
FirebaseVisionImage.fromFilePath(context, Uri.parse(imagePath));
            FirebaseVisionImage image =
FirebaseVisionImage.fromFilePath(context, Uri.fromFile(new File(imagePath)));
            analyze(image);
        } catch (IOException e) {
            Log.e(CLASS_LOG_TAG, "Failed to analyze image from path " +
imagePath + ". Cause: " + e.getMessage());
        }
    }


    public void analyze(FirebaseVisionImage image) {
        FirebaseVisionTextRecognizer detector;


        if (model == LOCAL_MODEL) {
            detector =
FirebaseVision.getInstance().getOnDeviceTextRecognizer();
            Log.i(CLASS_LOG_TAG, "Using local model");
        } else if (model == CLOUD_MODEL) {
            detector = FirebaseVision.getInstance().getCloudTextRecognizer();
            Log.i(CLASS_LOG_TAG, "Using cloud model");
        } else
            return;
```

```
        //Task<FirebaseVisionText> result = ..

        detector.processImage(image)

                .addOnSuccessListener(firebaseVisionText -> { // Task
completed successfully

                    Log.i(CLASS_LOG_TAG, "Recognition completed
successfully");

                    List<String> urls =
getUrls(firebaseVisionText.getText());

                    if (urls.isEmpty()) {

                        String msg = "Text recognized, however links were not
detected. Please, try again";

                        Toast.makeText(context, msg,
Toast.LENGTH_SHORT).show();

                        Log.i(CLASS_LOG_TAG, "Links were not detected. Make
sure it starts with 'https://', 'http://' or 'ftp://");

                    } else {

                        DataBase.addNewLink(urls);


                        for (String url : urls) {

                            Toast.makeText(context, url,
Toast.LENGTH_SHORT).show();

                            Log.i(CLASS_LOG_TAG, "URL detected: " + url);

                        }


                        if (urls.size() == 1) {

                            //String msg = "Url recognized. You can access it
from the main page";

                            //Toast.makeText(context, msg,
Toast.LENGTH_SHORT).show();

                            Log.i(CLASS_LOG_TAG, "Only one URL recognized.
Opening the browser");

                        } else {

                            //String msg = "Multiple urls recognized. You can
access them from the main page";

                            //Toast.makeText(context, msg,
Toast.LENGTH_SHORT).show();
```

```
                            Log.i(CLASS_LOG_TAG, "Multiple urls recognized.
Going back to MainActivity");
                        }
                        Thread thread = new Thread(() -> {
                            try {
                                Thread.sleep(1500);
                            } catch (InterruptedException e) {
                                Log.e(CLASS_LOG_TAG, e.getMessage());
                            }
                            activity.finish();
                        });
                        thread.start();

                    }
                })
                .addOnFailureListener(e -> { // Task failed with an exception
                    Log.i(CLASS_LOG_TAG, "Recognition failed");
                    Log.e(CLASS_LOG_TAG,
Objects.requireNonNull(e.getMessage()));

                    String msg = "Text recognition failed\n";
                    Toast.makeText(context, msg, Toast.LENGTH_LONG).show();
                });

    }

    /**
     * We call that on startup with hope the needed model will be downloaded
as soon as possible;
     * It is used to prevent: "Waiting for the text recognition model to be
downloaded. Please wait."
     * exception when recognizing.
     */
```

```
    public static void warmUp() {
        Log.i(CLASS_LOG_TAG, "warmUp() method call");
        Bitmap image = Bitmap.createBitmap(100, 100,
Bitmap.Config.ARGB_8888);
        image.eraseColor(android.graphics.Color.GREEN);
        FirebaseVisionImage firebaseVisionImage =
FirebaseVisionImage.fromBitmap(image);
        FirebaseVision.getInstance().getOnDeviceTextRecognizer()
                .processImage(firebaseVisionImage)
                .addOnSuccessListener(null)
                .addOnFailureListener(null);
    }

    public List<String> getUrls(String src) {
        Log.i(CLASS_LOG_TAG, "getUrls(String) method call");
        List<String> result = new ArrayList<>();
        String regex = "(http|ftp|https)://([\\w+?\\.\\w+])+([a-zA-Z0-
9\\~\\!\\@\\#\\$\\%\\^\\&\\*\\(\\)_\\-\\=\\+\\\\\\/\\?\\.\\:\\;\\'\\,]*)?";
        Pattern pattern = Pattern.compile(regex);
        Matcher matcher = pattern.matcher(src);
        while (matcher.find()) {
            String url = src.substring(matcher.start(), matcher.end());
            result.add(url);
        }
        return result;
    }

}
```

1.2. CameraActivity.java

```
package com.belkin.linker;


import androidx.annotation.NonNull;
import androidx.appcompat.app.ActionBar;
import androidx.appcompat.app.AppCompatActivity;
import androidx.camera.core.CameraX;
import androidx.camera.core.ImageCapture;
import androidx.camera.core.ImageCaptureConfig;
import androidx.camera.core.ImageProxy;
import androidx.camera.core.Preview;
import androidx.camera.core.PreviewConfig;
import androidx.core.app.ActivityCompat;
import androidx.core.content.ContextCompat;


import android.Manifest;
import android.app.Activity;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.graphics.Color;
import android.graphics.Matrix;
import android.os.Build;
import android.os.Bundle;
import android.util.Rational;
import android.util.Size;
import android.view.Surface;
import android.view.TextureView;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.ImageView;
import android.widget.Toast;
```

```
import com.esafirm.imagepicker.features.ImagePicker;
import com.esafirm.imagepicker.model.Image;

import java.util.ArrayList;
import java.util.List;



public class CameraActivity extends AppCompatActivity {


    private final int REQUEST_CODE_PERMISSIONS = 42;
    private final String[] REQUIRED_PERMISSIONS = {
Manifest.permission.CAMERA, Manifest.permission.WRITE_EXTERNAL_STORAGE };


    TextureView viewFinder;
    ImageView imgCaptureBtn;
    Button btnSelect;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_camera);


        //ActionBar
        ActionBar ab = getSupportActionBar();
        ab.hide();


        viewFinder = (TextureView) findViewById(R.id.view_finder);
        if (allPermissionsGranted()) {
            viewFinder.post(this::startCamera);
        } else {
            ActivityCompat.requestPermissions(this, REQUIRED_PERMISSIONS,
REQUEST_CODE_PERMISSIONS);
```

```
        }


        viewFinder.addOnLayoutChangeListener((View view, int i, int i1, int
i2, int i3, int i4, int i5, int i6, int i7) -> updateTransform());


        imgCaptureBtn = (ImageView) findViewById(R.id.imgCaptureBtn);


        btnSelect = (Button) findViewById(R.id.btnSelect);
        btnSelect.setOnClickListener(v -> {
            ArrayList<Image> images = new ArrayList<>();
            ImagePicker.create(this)
                    //.returnMode(ReturnMode.ALL) // set whether pick and /
or camera action should return immediate result or not.
                    .folderMode(true) // folder mode (false by default)
                    .toolbarFolderTitle("Folder") // folder selection title
                    .toolbarImageTitle("Tap to select") // image selection
title
                    .toolbarArrowColor(Color.BLACK) // Toolbar 'up' arrow
color
                    .includeVideo(false) // Show video on image picker
                    //.single() // single mode
                    .multi() // multi mode (default mode)
                    .limit(10) // max images can be selected (99 by default)
                    .showCamera(true) // show camera or not (true by default)
                    .imageDirectory("Camera") // directory name for captured
image  ("Camera" folder by default)
                    .origin(images) // original selected images, used in
multi mode
                    //.exclude(images) // exclude anything that in
image.getPath()
                    //.excludeFiles(files) // same as exclude but using
ArrayList<File>
                    //.theme(R.style.CustomImagePickerTheme) // must inherit
ef_BaseTheme. please refer to sample
                    .enableLog(true) // disabling log
```

```
                    .start(); // start image picker activity with request
code
        });
    }


    @Override
    protected void onActivityResult(int requestCode, final int resultCode,
Intent data) {
        if (ImagePicker.shouldHandle(requestCode, resultCode, data)) {
            List<Image> images = ImagePicker.getImages(data);
            for (Image image : images) {
                Analyzer analyzer = new Analyzer(getActivity());
                analyzer.analyze(image.getPath());
            }
        }
        super.onActivityResult(requestCode, resultCode, data);
    }


    @Override
    public void onRequestPermissionsResult(int requestCode, @NonNull String[]
permissions, @NonNull int[] grantResults) {
        if (requestCode == REQUEST_CODE_PERMISSIONS) {
            if (allPermissionsGranted()) {
                viewFinder.post(this::startCamera);
            } else {
                Toast.makeText(this, "Permissions not granted by the user.",
Toast.LENGTH_SHORT).show();
                finish();
            }
        }


    }
```

```
    private boolean allPermissionsGranted() {

        for (String permission : REQUIRED_PERMISSIONS) {

            if (ContextCompat.checkSelfPermission(getBaseContext(),
permission) == PackageManager.PERMISSION_DENIED)

                return false;

        }

        return true;

    }




    private void startCamera() {

        CameraX.unbindAll();


        Rational aspectRatio = new Rational(viewFinder.getWidth(),
viewFinder.getHeight());

        Size screen = new Size(viewFinder.getWidth(),
viewFinder.getHeight());


        PreviewConfig config = new
PreviewConfig.Builder().setTargetAspectRatio(aspectRatio).setTargetResolution
(screen).build();

        Preview preview = new Preview(config);


        preview.setOnPreviewOutputUpdateListener(output -> {

            ViewGroup parent = (ViewGroup) viewFinder.getParent();

            parent.removeView(viewFinder);

            parent.addView(viewFinder, 0);

            viewFinder.setSurfaceTexture(output.getSurfaceTexture());

            updateTransform();

        });


        ImageCaptureConfig imageCaptureConfig = new
ImageCaptureConfig.Builder().setCaptureMode(ImageCapture.CaptureMode.MAX_QUAL
ITY)
```

```
.setTargetRotation(getWindowManager().getDefaultDisplay().getRotation()).buil
d();
        final ImageCapture imgCapture = new ImageCapture(imageCaptureConfig);
        imgCaptureBtn.setOnClickListener(v -> {
            imgCaptureBtn.setEnabled(false);
            imgCapture.takePicture(new ImageCapture.OnImageCapturedListener()
{
                @Override
                public void onCaptureSuccess(ImageProxy image, int
rotationDegrees) {
                    Analyzer analyzer = new Analyzer(getActivity());
                    analyzer.analyze(image, rotationDegrees);
                    imgCaptureBtn.setEnabled(true);
                    //if (error with download ask user to update it's google
play services)
                }
            });

        });

        CameraX.bindToLifecycle(this, preview, imgCapture);
    }

    private Activity getActivity() {
        return this;
    }

    private void updateTransform() {
        Matrix mx = new Matrix();
        float w = viewFinder.getMeasuredWidth();
        float h = viewFinder.getMeasuredHeight();
```

```
        float cX = w / 2f;
        float cY = h / 2f;
        int rotationDgr;
        int rotation = (int) viewFinder.getRotation();

        switch (rotation) {
            case Surface.ROTATION_0:
                rotationDgr = 0;
                break;
            case Surface.ROTATION_90:
                rotationDgr = 90;
                break;
            case Surface.ROTATION_180:
                rotationDgr = 180;
                break;
            case Surface.ROTATION_270:
                rotationDgr = 270;
                break;
            default:
                return;
        }

        if (isEmulator())
            rotationDgr -= 90;

        mx.postRotate((float) rotationDgr, cX, cY);
        viewFinder.setTransform(mx);

    }

    public static boolean isEmulator() {
        return Build.FINGERPRINT.startsWith("generic")
```

```
                || Build.FINGERPRINT.startsWith("unknown")
                || Build.MODEL.contains("google_sdk")
                || Build.MODEL.contains("Emulator")
                || Build.MODEL.contains("Android SDK built for x86")
                || Build.MANUFACTURER.contains("Genymotion")
                || (Build.BRAND.startsWith("generic") &&
Build.DEVICE.startsWith("generic"))
                || "google_sdk".equals(Build.PRODUCT);
    }
}
```

1.3. DataBase.java

```
package com.belkin.linker;

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;

class DataBase {
    final static String CLASS_LOG_TAG = "DataBase";

    static private DBHelper dbHelper;
    static private SQLiteDatabase db;
    static private List<Link> links;
    static private LinkToListItemAdapter adapter;
    static private Context context;
```

```
    private DataBase() {}

    static void setContext(Context mContext) {
        Log.i(CLASS_LOG_TAG, "setContext(Context) method call");
        context = mContext;
        if(links == null)
            links = new ArrayList<>();
    }

    static void setAdapter(LinkToListItemAdapter mAdapter) {
        adapter = mAdapter;
    }

    static void readAllDataBase() {
        Log.i(CLASS_LOG_TAG, "readAllDataBase() method call");
        if (openDatabase()) {
            db = dbHelper.getWritableDatabase();
            Cursor c = db.query(DBHelper.TABLE_NAME, null, null, null, null,
null, null);
            if (c.moveToFirst()) {
                // определяем номера столбцов по имени в выборке
                int idIndex = c.getColumnIndex("id");
                int urlIndex = c.getColumnIndex("url");
                int hostIndex = c.getColumnIndex("host");
                int headerIndex = c.getColumnIndex("header");
                int datetimeIndex = c.getColumnIndex("datetime");
                int imageUrlIndex = c.getColumnIndex("imageUrl");
                do {
                    // получаем значения по номерам столбцов и пишем все в
лог
                    int id = c.getInt(idIndex);
                    String url = c.getString(urlIndex);
```

```
                    String host = c.getString(hostIndex);
                    String header = c.getString(headerIndex);
                    String datetime = c.getString(datetimeIndex);
                    String imageUrl = c.getString(imageUrlIndex);
                    Link link = new Link(id, url, host, header, imageUrl,
datetime);
                    links.add(link);
                    Log.i(CLASS_LOG_TAG, "Read from database " +
DBHelper.DB_NAME + " from table " + DBHelper.TABLE_NAME + ": " +
link.toString());
                } while (c.moveToNext());
            }
            dbHelper.close();
        }
    }

    static void writeToDataBase(List<Link> links) {
        Log.i(CLASS_LOG_TAG, "writeToDataBase(List<Link>) method call");
        if (openDatabase()) {
            db = dbHelper.getWritableDatabase();
            for (Link link : links) {
                dbWrite(link);
            }
            db.close();
        }
    }

    static void writeToDataBase(Link link) {
        Log.i(CLASS_LOG_TAG, "writeToDataBase(Link) method call");
        if (openDatabase()) {
            db = dbHelper.getWritableDatabase();
            dbWrite(link);
            db.close();
```

```
        }
    }

    private static long dbWrite(Link link) {
        Log.i(CLASS_LOG_TAG, "insertToDataBase(Link) method call");
        if (openDatabase()) {
            ContentValues cv = new ContentValues();
            cv.put("url", link.getUrl());
            cv.put("host", link.getHost());
            cv.put("header", link.getHeader());
            cv.put("datetime", link.getDatetime());
            cv.put("imageUrl", link.getImageUrl());
            link.setId(db.insert(DBHelper.TABLE_NAME, null, cv));

            Log.i(CLASS_LOG_TAG, "Write to database " + DBHelper.DB_NAME + "
to table " + DBHelper.TABLE_NAME + ": " + link.toString());

            return link.getId();
        }
        return -1;
    }

    static void deleteFromDataBase(List<Link> links) {
        Log.i(CLASS_LOG_TAG, "deleteFromDataBase(List<Link>) method call");
        if (openDatabase()) {
            for (Link link : links) {
                deleteFromDataBase(link.getId());
            }
        }
    }

    static void deleteFromDataBase(Link link) {
```

```
        Log.i(CLASS_LOG_TAG, "deleteFromDataBase(Link) method call");
        if (openDatabase()) {
            deleteFromDataBase(link.getId());
        }
    }

    static private void deleteFromDataBase(long id) {
        Log.i(CLASS_LOG_TAG, "deleteFromDataBase(long) method call");
        if (openDatabase()) {
            int delCount = db.delete(DBHelper.TABLE_NAME, "id=?", new
String[]{String.valueOf(id)});
            if (delCount == 1) {
                Log.i(CLASS_LOG_TAG, "Delete from database " +
DBHelper.DB_NAME + " from table " + DBHelper.TABLE_NAME + ": row id = " +
id);
            }
            else {
                Log.i(CLASS_LOG_TAG, "Cannot delete from database " +
DBHelper.DB_NAME + " from table " + DBHelper.TABLE_NAME + ": row id = " + id
+ " not found");
            }

            try {

            }
            catch (Exception e) {
                Log.d("DEBUG", e.getMessage());
            }

        }
    }

    private static boolean openDatabase() {
```

```
        Log.i(CLASS_LOG_TAG, "isDbHelperSet() method call");
        if (dbHelper == null) {
            Log.i(CLASS_LOG_TAG, "DBHelper is not set");
            Log.i(CLASS_LOG_TAG, "Trying to get it from context");
            Log.e(CLASS_LOG_TAG, "DBHelper is not set. Context is not
provided. Please, provide context with setContext() method");

            if (context == null) {
                Log.i(CLASS_LOG_TAG, "Context is not set");
                Log.e(CLASS_LOG_TAG, "Context is not provided. Please,
provide context with setContext() method");
                return false;
            }
            else {
                dbHelper = new DBHelper(context);
                db = dbHelper.getWritableDatabase();
                Log.i(CLASS_LOG_TAG, "DBHelper is set");
                Log.i(CLASS_LOG_TAG, "Database is set");
                return true;
            }
        }
        else {
            db = dbHelper.getWritableDatabase();
            Log.i(CLASS_LOG_TAG, "Database is set");
            return true;
        }
    }

    static List<Link> getData() {
        Log.i(CLASS_LOG_TAG, "getData() method call");
        return links;
    }
```

```
    static void addNewLink(List<String> urls) {
        Log.i(CLASS_LOG_TAG, "addNewLink(List<String>) method call");
        for (String url : urls) {
            addNewLink(url);
        }
    }

    static void addNewLink(String url) {
        Log.i(CLASS_LOG_TAG, "addNewLink(String) method call");
        Link link = new Link(url, adapter, links.size());
        links.add(link);
        adapter.notifyDataSetChanged();

    }

    static private class DBHelper extends SQLiteOpenHelper  {
        final private static String CLASS_LOG_TAG = DataBase.CLASS_LOG_TAG +
".DBHelper";

        final static String TABLE_NAME = "mytable";
        final static String DB_NAME = "myDB";

        DBHelper(Context context) {
            super(context, DB_NAME, null, 1);
        }

        @Override
        public void onCreate(SQLiteDatabase db) {
            Log.i(CLASS_LOG_TAG, "onCreate(SQLiteDatabase) method call");
            db.execSQL("CREATE TABLE " + TABLE_NAME + " " +
```

```
                    "(" +
                    "id integer primary key autoincrement," +
                    "url text," +
                    "host text," +
                    "header text," +
                    "datetime text," +
                    "imageUrl text" +
                    ")");
        }

        @Override
        public void onUpgrade(SQLiteDatabase db, int oldVersion, int
newVersion) {
            Log.i(CLASS_LOG_TAG, "onUpgrade(SQLiteDatabase, int, int) method
call");
        }
    }
}
```

1.4. Link.java

```
package com.belkin.linker;

import android.os.AsyncTask;
import android.util.Log;

import org.jsoup.Jsoup;
import org.jsoup.nodes.Document;
import org.jsoup.select.Elements;

import java.io.IOException;
import java.net.URI;
import java.net.URISyntaxException;
import java.text.SimpleDateFormat;
```

```
import java.util.Calendar;

public class Link {
    private static String CLASS_LOG_TAG = "Link";

    private final String HEADER_PLACEHOLDER = "Cannot Open Page";

    private String header;
    private String host;
    private String datetime;
    private String imageUrl;
    private String url;
    private long id;


    Link(String url, LinkToListItemAdapter adapter, int position) {
        this.url = url;
        new DownloadTask(adapter, position).execute();
    }

    private class DownloadTask extends AsyncTask<Void, Void, Void> {
        private LinkToListItemAdapter adapter;
        private int pos;

        DownloadTask(LinkToListItemAdapter adapter, int pos) {
            this.adapter = adapter;
            this.pos = pos;
        }

        @Override
        protected void onPreExecute() {
            super.onPreExecute();
```

```
            setLoading();
            adapter.notifyItemChanged(pos);
        }

        @Override
        protected Void doInBackground(Void... params) {
            setHost();
            setDatetime();
            setImageAndHeader();
            return null;
        }

        @Override
        protected void onProgressUpdate(Void... values) {
            super.onProgressUpdate(values);
            adapter.animateLoading(pos);
        }

        @Override
        protected void onPostExecute(Void result) {
            super.onPostExecute(result);
            adapter.notifyItemChanged(pos);
            DataBase.writeToDataBase(getInstance());
        }
    }

    private void setLoading() {
        header = "Loading..";
        datetime = "";
        host = "";
    }
```

```
    private Link getInstance() {
        return this;
    }

    Link(long id, String url, String host, String header, String imageUrl,
String datetime) {
        this.id = id;
        this.url = url;
        this.host = host;
        this.header = header;
        this.imageUrl = imageUrl;
        this.datetime = datetime;
    }

    //getters
    String getImageUrl() {
        return imageUrl;
    }

    String getHeader() {
        return header;
    }

    String getHost() {
        return host;
    }

    String getDatetime() {
        return datetime;
    }

    String getUrl() {
```

```
        return url;
    }

    long getId() {
        return id;
    }



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


    private void setHeader(Document doc) {
        Log.i(CLASS_LOG_TAG, "setHeader(Document) method called");


        //1st priority: <meta property="og:title" content="X" /> where X is
page title
        Elements els = doc.select("meta").select("[property=og:title]");
        if (els.size() != 0) {
            header = els.get(0).attr("content");
            Log.i(CLASS_LOG_TAG, "setHeader() found 1st priority header");
            return;
        }


        //2nd priority: <title>X</title> where X is page title
        els = doc.select("title");
        if (els.size() != 0) {
            header = els.get(0).text();
            Log.i(CLASS_LOG_TAG, "setHeader() found 2nd priority header");
            return;
        }
```

```
        //3rd priority: <h1>X</h1> where X is page title
        els = doc.select("h1");
        if (els.size() != 0) {
            header = els.get(0).text();
            Log.i(CLASS_LOG_TAG, "setHeader() found 3rd priority header");
            return;
        }

        //4th priority: host
        header = getHost();
        Log.i(CLASS_LOG_TAG, "setHeader() did not find any header, using host
name instead");
    }

    private void setImage(Document doc) {
        Log.i(CLASS_LOG_TAG, "setImage(Document) method called");

        //1st priority: <meta property="og:image" content="X" /> where X is
image url and X can't contain "favicon" char sequence
        Elements els =
doc.select("meta").select("[property=og:image]").select("[content~=^((?!favic
on).)*$]");
        if (els.size() != 0) {
            imageUrl = els.get(0).attr("content");
            Log.i(CLASS_LOG_TAG, "setImage() found 1st priority image");
            return;
        }

        //2nd priority: <img src="X"/> where X is image url and X can't
contain "favicon" char sequence
        els =
doc.select("img").select("[src~=.(png|jpe?g)]").select("[src~=^((?!favicon).)
*$]");
```

```
        if (els.size() != 0) {
            imageUrl = els.get(0).attr("src");
            Log.i(CLASS_LOG_TAG, "setImage() found 2nd priority image");
            return;
        }

        //3rd priority: <link href="X"/> where X is image url and X can't
contain "favicon" char sequence
        els =
doc.select("link").select("[href~=.(png|jpe?g)]").select("[href~=^((?!favicon
).)*$]");
        if (els.size() != 0) {
            imageUrl = els.get(0).attr("href");
            Log.i(CLASS_LOG_TAG, "setImage() found 3rd priority image");
            return;
        }

        //4th priority: <meta property="og:image" content="X" /> where X is
image url and X can contain "favicon" char sequence
        els = doc.select("meta").select("[property=og:image]");
        if (els.size() != 0) {
            imageUrl = els.get(0).attr("content");
            Log.i(CLASS_LOG_TAG, "setImage() found 4th priority image");
            return;
        }

        //5th priority: <img src="X"/> where X is image url and X can't
contain "favicon" char sequence
        els = doc.select("img").select("[src~=.(png|jpe?g)]");
        if (els.size() != 0) {
            imageUrl = els.get(0).attr("src");
            Log.i(CLASS_LOG_TAG, "setImage() found 5th priority image");
            return;
```

```
        }

        //6th priority: <link href="X"/> where X is image url and X can't
contain "favicon" char sequence
        els = doc.select("link").select("[href~=.(png|jpe?g)]");
        if (els.size() != 0) {
            imageUrl = els.get(0).attr("href");
            Log.i(CLASS_LOG_TAG, "setImage() found 6th priority image");
            return;
        }

        Log.i(CLASS_LOG_TAG, "setImage() did not find any image, using
placeholder");
    }



    private void setImageAndHeader() {
        Log.i(CLASS_LOG_TAG, "setImageAndHeader() method called");

        header = HEADER_PLACEHOLDER;
        if (url != null) {
            Log.i(CLASS_LOG_TAG, "Reading web page from URL: " + url);
            try {
                Document doc = Jsoup.connect(url)
                        .userAgent("Chrome/4.0.249.0 Safari/532.5")
                        .referrer("http://www.google.com")
                        .get();

                setImage(doc);
                setHeader(doc);
                Log.i(CLASS_LOG_TAG, "Header is set: " + header);
                Log.i(CLASS_LOG_TAG, "Image url is set: " + imageUrl);
```

```
            } catch (IOException e) {
                Log.e(CLASS_LOG_TAG, "Error parsing image and header from
url. " + e.getMessage());
            }

        } else {
            Log.e(CLASS_LOG_TAG, "Error parsing image and header from url.
Url is null");
        }

    }

    private void setHost() {
        try {
            URI uri = new URI(url);
            String domain = uri.getHost();
            host = domain.startsWith("www.") ? domain.substring(4) : domain;
        } catch (URISyntaxException e) {
            e.printStackTrace();
        }
    }

    private void setDatetime() {
        datetime = new SimpleDateFormat("dd/MM/yyyy
HH:mm:ss").format(Calendar.getInstance().getTime());
    }

    @Override
    public String toString() {
        return "Id = " + id + " " +
                "URL = " + url + " " +
                "Host = " + host + " " +
                "Header = " + header + " " +
```

```
                "Datetime = " + datetime;
    }
}
```

1.5. LinkToListItemAdapter.java

```
package com.belkin.linker;

import android.content.Context;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ImageView;
import android.widget.TextView;

import androidx.annotation.NonNull;
import androidx.constraintlayout.widget.ConstraintLayout;
import androidx.recyclerview.widget.RecyclerView;

import com.squareup.picasso.Picasso;

import java.util.List;

public class LinkToListItemAdapter extends
RecyclerView.Adapter<LinkToListItemAdapter.ViewHolder> {

    private LayoutInflater inflater;
    private List<Link> links;
    private RecyclerItemTouchHelper.RecyclerItemTouchHelperListener listener;
    private Context context;

    LinkToListItemAdapter(Context context, List<Link> links,
RecyclerItemTouchHelper.RecyclerItemTouchHelperListener listener) {
        this.links = links;
```

```
        this.inflater = LayoutInflater.from(context);
        this.listener = listener;
        this.context = context;
    }

    @NonNull
    @Override
    public LinkToListItemAdapter.ViewHolder onCreateViewHolder(@NonNull
ViewGroup parent, int viewType) {
        View view = inflater.inflate(R.layout.list_item, parent, false);
        return new ViewHolder(view);
    }

    @Override
    public void onBindViewHolder(LinkToListItemAdapter.ViewHolder holder, int
position) {
        Link link = links.get(position);
        String imageUrl = link.getImageUrl();
        if (imageUrl != null) {
            Picasso.with(context).load(imageUrl).into(holder.imageView);
            holder.imageView.setVisibility(View.VISIBLE);
        } else {
            holder.imageView.setVisibility(View.INVISIBLE);
        }
        holder.datetimeView.setText(link.getDatetime());
        holder.hostView.setText(link.getHost());
        holder.headerView.setText(link.getHeader());
    }

    @Override
    public int getItemCount() {
        return links.size();
```

```
    }

    void removeItem(int position) {
        links.remove(position);
        notifyItemRemoved(position);
        // NOTE: don't call notifyDataSetChanged()
    }

    void restoreItem(Link item, int position) {
        links.add(position, item);
        notifyItemInserted(position);
    }

    void animateLoading(int pos) {
        //todo:

    }

    class ViewHolder extends RecyclerView.ViewHolder implements
View.OnClickListener {
        final ImageView imageView;
        final TextView datetimeView, hostView, headerView;
        final ConstraintLayout viewBackgroundDelete;
        final ConstraintLayout viewBackgroundShare;
        final ConstraintLayout viewForeground;

        ViewHolder(View view) {
            super(view);
            imageView = (ImageView) view.findViewById(R.id.image);
            hostView = (TextView) view.findViewById(R.id.host);
            headerView = (TextView) view.findViewById(R.id.header);
            datetimeView = (TextView) view.findViewById(R.id.datetime);
```

```
            viewBackgroundDelete = (ConstraintLayout)
view.findViewById(R.id.item_background_delete);

            viewBackgroundShare = (ConstraintLayout)
view.findViewById(R.id.item_background_share);

            viewForeground = (ConstraintLayout)
view.findViewById(R.id.item_foreground);

            viewForeground.setOnClickListener(this);

        }


        @Override

        public void onClick(View v) {

            listener.onClicked(this);

        }

    }



}
```

1.6. MainActivity.java

```
package com.belkin.linker;


import androidx.annotation.NonNull;

import androidx.appcompat.app.AppCompatActivity;

import androidx.constraintlayout.widget.ConstraintLayout;

import androidx.core.app.ActivityCompat;

import androidx.core.content.ContextCompat;

import androidx.recyclerview.widget.DefaultItemAnimator;

import androidx.recyclerview.widget.ItemTouchHelper;

import androidx.recyclerview.widget.RecyclerView;


import android.Manifest;

import android.content.Intent;

import android.content.pm.PackageManager;

import android.graphics.Color;
```

```
import android.net.Uri;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.ImageButton;
import android.widget.SearchView;
import android.widget.Toast;

import com.google.android.material.snackbar.Snackbar;

import java.util.List;

public class MainActivity extends AppCompatActivity {

    final static String CLASS_LOG_TAG = "MainActivity";

    private final int REQUEST_CODE_PERMISSIONS = 42;
    private final String[] REQUIRED_PERMISSIONS = {
            Manifest.permission.CAMERA,
            Manifest.permission.READ_EXTERNAL_STORAGE,
            Manifest.permission.WRITE_EXTERNAL_STORAGE,
            Manifest.permission.INTERNET,
    };


    List<Link> displayedLinks;
    LinkToListItemAdapter adapter;
    ConstraintLayout rootLayout;
    ItemTouchHelper.SimpleCallback itemTouchHelperCallback;
    RecyclerView recyclerView;

    @Override
```

```
    public void onRequestPermissionsResult(int requestCode, @NonNull String[]
permissions, @NonNull int[] grantResults) {
        if (requestCode == REQUEST_CODE_PERMISSIONS) {
            if (!allPermissionsGranted()) {
                Toast.makeText(this, "Permissions not granted by the user.",
Toast.LENGTH_SHORT).show();
                finish();
            }
        }
    }

    private boolean allPermissionsGranted() {
        for (String permission : REQUIRED_PERMISSIONS) {
            if (ContextCompat.checkSelfPermission(getBaseContext(),
permission) == PackageManager.PERMISSION_DENIED)
                return false;
        }
        return true;
    }



    @Override
    protected void onRestart() {
        super.onRestart();
        onListChanged();
    }


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        Log.i(CLASS_LOG_TAG, "onCreate(Bundle) method call");
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
```

```
        //download ML model if it's not downloaded yet
        Analyzer.warmUp();



        //DataBase
        DataBase.setContext(getBaseContext());
        DataBase.readAllDataBase();
        displayedLinks = DataBase.getData();
        MyRecyclerItemTouchHelperListener listener = new
MyRecyclerItemTouchHelperListener();
        adapter = new LinkToListItemAdapter(this, displayedLinks, listener);
        DataBase.setAdapter(adapter);


        //ActionBar
        getSupportActionBar().hide();


        //Root Layout
        rootLayout = (ConstraintLayout) findViewById(R.id.main_root);


        //RecycleView
        recyclerView = (RecyclerView) findViewById(R.id.list);
        recyclerView.setAdapter(adapter);
        recyclerView.setItemAnimator(new DefaultItemAnimator());
        //recyclerView.addItemDecoration(new DividerItemDecoration(this,
DividerItemDecoration.VERTICAL));
        itemTouchHelperCallback = new RecyclerItemTouchHelper(0,
ItemTouchHelper.LEFT | ItemTouchHelper.RIGHT, listener);
        new
ItemTouchHelper(itemTouchHelperCallback).attachToRecyclerView(recyclerView);


        //scan btn
        ImageButton scanBtn = (ImageButton) findViewById(R.id.scanBtn);
        scanBtn.setOnClickListener(v -> {
```

```
            Intent intent = new Intent(getBaseContext(),
CameraActivity.class);
            startActivity(intent);
            Log.i(CLASS_LOG_TAG, "CameraActivity started");
        });


        //todo: search view
        //searchView
        SearchView searchView = (SearchView) findViewById(R.id.searchView);


        onListChanged();



        if (!allPermissionsGranted()) {
            ActivityCompat.requestPermissions(this, REQUIRED_PERMISSIONS,
REQUEST_CODE_PERMISSIONS);
        }
    }



    public void onListChanged() {
        Log.i(CLASS_LOG_TAG, "onListChanged() method call");
        if (adapter.getItemCount() == 0) {
            Log.i(CLASS_LOG_TAG, "List is empty. Showing info icon and
text");
            findViewById(R.id.empty_list_image).setVisibility(View.VISIBLE);

findViewById(R.id.empty_list_description1).setVisibility(View.VISIBLE);

findViewById(R.id.empty_list_description2).setVisibility(View.VISIBLE);
        } else {
            Log.i(CLASS_LOG_TAG, "List is not empty. Hiding info icon and
text");
```

```
findViewById(R.id.empty_list_image).setVisibility(View.INVISIBLE);

findViewById(R.id.empty_list_description1).setVisibility(View.INVISIBLE);

findViewById(R.id.empty_list_description2).setVisibility(View.INVISIBLE);
        }
    }


    class MyRecyclerItemTouchHelperListener implements
RecyclerItemTouchHelper.RecyclerItemTouchHelperListener {
        final static String CLASS_LOG_TAG = MainActivity.CLASS_LOG_TAG +
".TchListnr";


        @Override
        public void onSwiped(RecyclerView.ViewHolder viewHolder, int
direction, int position) {
            Log.i(CLASS_LOG_TAG, "onSwiped() method call");
            if (viewHolder instanceof LinkToListItemAdapter.ViewHolder) {
                switch (direction) {
                    case ItemTouchHelper.LEFT:
                        Log.i(CLASS_LOG_TAG, "Left swipe performed");
                        // get the removed item name to display it in snack
bar
                        String name =
displayedLinks.get(viewHolder.getAdapterPosition()).getHost();


                        // backup of removed item for undo purpose
                        final Link deletedItem =
displayedLinks.get(viewHolder.getAdapterPosition());
                        final int deletedIndex =
viewHolder.getAdapterPosition();


                        // remove the item from recycler view
                        adapter.removeItem(viewHolder.getAdapterPosition());
```

```
                        DataBase.deleteFromDataBase(deletedItem);
                        onListChanged();
                        Log.i(CLASS_LOG_TAG, "Item deleted: " +
deletedItem.toString());


                        // showing snack bar with Undo option
                        Snackbar snackbar = Snackbar.make(rootLayout, name +
" removed from list!", Snackbar.LENGTH_LONG);
                        snackbar.setAction("UNDO", view -> {
                            // undo is selected, restore the deleted item
                            adapter.restoreItem(deletedItem, deletedIndex);
                            DataBase.writeToDataBase(deletedItem);
                            onListChanged();
                            Log.i(CLASS_LOG_TAG, "Item restored: " +
deletedItem.toString());
                        });
                        snackbar.setActionTextColor(Color.YELLOW);


                        snackbar.show();
                        break;
                    case ItemTouchHelper.RIGHT:
                        Log.i(CLASS_LOG_TAG, "Right swipe performed");
                        String url =
displayedLinks.get(viewHolder.getAdapterPosition()).getUrl();
                        Intent share = new Intent(Intent.ACTION_SEND);
                        share.setType("text/plain");
                        share.putExtra(Intent.EXTRA_TEXT, url);
                        startActivity(Intent.createChooser(share, "Share
link"));
                        Log.i(CLASS_LOG_TAG, "Share activity started");


                        final Link deletedItem1 =
displayedLinks.get(viewHolder.getAdapterPosition());
```

```
                        final int deletedIndex1 =
viewHolder.getAdapterPosition();
                        adapter.removeItem(viewHolder.getAdapterPosition());
                        adapter.restoreItem(deletedItem1, deletedIndex1);


                        break;
                }
            }
        }

        @Override
        public void onClicked(RecyclerView.ViewHolder viewHolder) {
            Log.i(CLASS_LOG_TAG, "onClicked() method call");
            String url =
displayedLinks.get(viewHolder.getAdapterPosition()).getUrl();
            Intent browserIntent = new Intent(Intent.ACTION_VIEW,
Uri.parse(url));
            startActivity(browserIntent);
            Log.i(CLASS_LOG_TAG, "Browser activity started");
        }
    }


}
```

1.7. RecyclerItemTouchHelper.java

```
package com.belkin.linker;


import android.graphics.Canvas;
import android.util.Log;
import android.view.View;


import androidx.recyclerview.widget.ItemTouchHelper;
```

```
import androidx.recyclerview.widget.RecyclerView;


public class RecyclerItemTouchHelper extends ItemTouchHelper.SimpleCallback {


    private final static String CLASS_LOG_TAG = "RecyclerItemTouchHelper";


    private RecyclerItemTouchHelperListener listener;


    RecyclerItemTouchHelper(int dragDirs, int swipeDirs,
RecyclerItemTouchHelperListener listener) {
        super(dragDirs, swipeDirs);
        this.listener = listener;
    }


    @Override
    public boolean onMove(RecyclerView recyclerView, RecyclerView.ViewHolder
viewHolder, RecyclerView.ViewHolder target) {
        return true;
    }


    @Override
    public void onSelectedChanged(RecyclerView.ViewHolder viewHolder, int
actionState) {
        if (viewHolder != null) {
            final View foregroundView = ((LinkToListItemAdapter.ViewHolder)
viewHolder).viewForeground;
            Log.d(CLASS_LOG_TAG, "onSelectedChanged");
            getDefaultUIUtil().onSelected(foregroundView);
        }
    }


    @Override
```

```
    public void onChildDrawOver(Canvas c, RecyclerView recyclerView,
RecyclerView.ViewHolder viewHolder, float dX, float dY,
                                int actionState, boolean isCurrentlyActive) {
        final View foregroundView = ((LinkToListItemAdapter.ViewHolder)
viewHolder).viewForeground;
        Log.d(CLASS_LOG_TAG, "onChildDrawOver");
        getDefaultUIUtil().onDrawOver(c, recyclerView, foregroundView, dX,
dY, actionState, isCurrentlyActive);
    }

    @Override
    public void clearView(RecyclerView recyclerView, RecyclerView.ViewHolder
viewHolder) {
        final View foregroundView = ((LinkToListItemAdapter.ViewHolder)
viewHolder).viewForeground;
        Log.d(CLASS_LOG_TAG, "clearView");
        getDefaultUIUtil().clearView(foregroundView);
    }

    @Override
    public void onChildDraw(Canvas c, RecyclerView recyclerView,
RecyclerView.ViewHolder viewHolder, float dX, float dY,
                            int actionState, boolean isCurrentlyActive) {
        final View foregroundView = ((LinkToListItemAdapter.ViewHolder)
viewHolder).viewForeground;
        final View backgroundDeleteView = ((LinkToListItemAdapter.ViewHolder)
viewHolder).viewBackgroundDelete;
        final View backgroundShareView = ((LinkToListItemAdapter.ViewHolder)
viewHolder).viewBackgroundShare;
        if (dX < 0) {
            backgroundDeleteView.setVisibility(View.VISIBLE);
            backgroundShareView.setVisibility(View.INVISIBLE);
        } else {
            backgroundDeleteView.setVisibility(View.INVISIBLE);
            backgroundShareView.setVisibility(View.VISIBLE);
```

```
        }

        Log.d(CLASS_LOG_TAG, "onChildDraw");

        getDefaultUIUtil().onDraw(c, recyclerView, foregroundView, dX, dY,
actionState, isCurrentlyActive);

    }


    @Override

    public void onSwiped(RecyclerView.ViewHolder viewHolder, int direction) {

        Log.d(CLASS_LOG_TAG, "onSwiped");

        listener.onSwiped(viewHolder, direction,
viewHolder.getAdapterPosition());

    }


    @Override

    public int convertToAbsoluteDirection(int flags, int layoutDirection) {

        return super.convertToAbsoluteDirection(flags, layoutDirection);

    }


    public interface RecyclerItemTouchHelperListener {

        void onSwiped(RecyclerView.ViewHolder viewHolder, int direction, int
position);

        void onClicked(RecyclerView.ViewHolder viewHolder);

    }

}
```

ПРИЛОЖЕНИЕ

Код программы также доступен в репозитории на GitHub по ссылке:
https://github.com/belkin1667/linker