Izrada Android aplikacije za optimalno raspoređivanje fotonaponskih panela

Butković, Antonio

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Stručni studij

ANDROID APLIKACIJA ZA OPTIMALNO RASPOREĐIVANJE FOTONAPONSKIH PANELA

Završni rad

Antonio Butković

Osijek, 2018.

SADRŽAJ

1. UVOD

Android aplikacija za optimalno raspoređivanje fotonaponskih panela ima u cilj učinkovitije i efikasnije obavljati dosadašnji posao raspoređivanja panela. Za razliku od klasičnog postavljanja solarnih panela, Android aplikacija za optimalno raspoređivanje fotonaponskih panela nudi brži i pregledniji način obavljanja spomenutog posla. Kako bi se izbacile gomile tablica na kojima su priloženi položaji određenih panela, pomoću aplikacije potrebno je samo skenirati kod i pročitati položaj koji aplikacija prikaže. Štednja vremena je ogromna iz razloga što nije potrebno tražiti u tablici bar kod svakog panela, a zatim iščitati položaj na koji ga je potrebno smjestiti.

U drugom dijelu rada nalazi se uvod u tehnologije koje su korištene za izradu aplikacija, neke njihove osnovne karakteristike te primjeri. Treći dio rada opisuje funkcioniranje, izgled, strukturu aplikacije te bazu podataka uz predočenje programskog koda za bolje shvaćanje određenog dijela. Zaključak sadrži osvrt na sveukupni rad, izgled i izradu aplikacije, kao i ostvarene rezultate.

1.1. Zadatak završnog rada

Zadatak ovog završnog rada je izraditi Android aplikaciju koja će optimalno raspoređivati fotonaponske panele. U teorijskom dijelu rada treba objasniti sve alate i programe koji su potrebni za izradu praktičnog dijela završnog rada i pojasniti samu funkciju istih.

2. KORIŠTENE TEHNOLOGIJE

2.1. Android (operacijski sustav)

Google Android je prvi otvoreni operacijski sustav za mobilne uređaje (mobilni telefoni, tableti, netbook računala, Google TV) pokrenut od strane Google Inc. i vođen od strane Open Handset Alliance - grupe koja danas broji preko 80 tehnoloških kompanija između kojih se nalaze T-Mobile, HTC, Intel, Motorola, Qualcomm, i drugi, čiji je cilj ubrzati inovacije na području mobilnih operacijskih sustava, a samim time ponuditi krajnjim kupcima bogatije, jeftinije i bolje iskustvo uporabe, prema [1].

Android platforma prilagođena je uporabi uređaja s većim zaslonima. Konstantnim izlaskom novih verzija Androida mnoge značajke se mijenjaju na bolje. Android je u vrlo kratkom razdoblju svojeg postojanja postigao vrlo veliku popularnost i širok spektar korisnika. Prema slikama 2.1. i 2.2. prikazane su starija i novija verzija Androida.

Sl. 2.1. Android 2.2 Gingerbread, prema [2].

Sl. 2.2. Android 7.0 Nougat, prema [3].

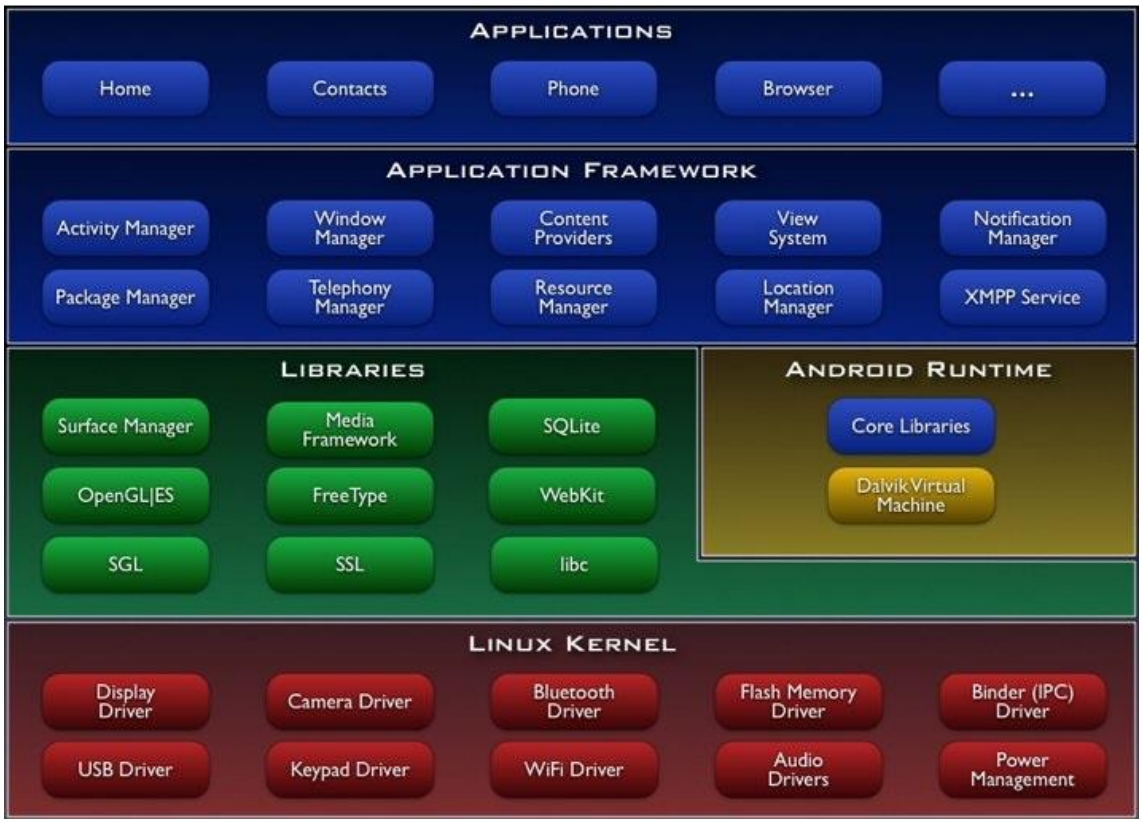
2.2. Arhitektura

Prema [1] Android je zasnovan na jezgri Linux 2.6 i napisan u C/C++ programskom jeziku. Većina aplikacija pisana je u Java programskom jeziku rabeći Android razvojno programsko okruženje (engl. *Android Software Development Kit*, SDK).

Arhitekturu Androida moguće je promatrati kao jedan programski sloj koji sadrži 5 cjelina smještenih u 4 sloja:

- Jezgra Linuxa OS-a (engl. *Linux Kernel*)
- Biblioteke (engl. *Libraries*), Android radno okruženje (engl. *Android RunTime)*
- Aplikacijski okvir (engl. *Application Framework*)
- Aplikacije

Prema slici 2.3. vidljive su ranije spomenute razine i slojevi.



Sl. 2.3. Arhitektura Android operativnog sustava, prema [4].

Na samom dnu nalazi se Linux 2.6 jezgra koja sadrži sve pogonske programe (engl. *driver*) za uređaje (zaslon, kamera, audio, pogonski program za upravljanje napajanjem (engl. *Power management),…)* od kojih je potrebno izdvojiti dva najvažnija: pogonski program za međuprocesnu komunikaciju (engl. *Inter-process communication*, IPC) koji služi za izmjenu podataka između različitih procesa ili niti unutar procesa te pogonski program za upravljanje napajanjem.

Iznad Linux jezgre nalaze se knjižnice koje su pisane u C/C++ programskom jeziku (SQLite, WebKit, graphic i media).

Zatim, Android Runtime koji služi za pokretanje aplikacija, a sastoji se od dvije važne komponente: jezgrenih knjižnica (engl. *Core libraries*) i Dalvik virtualnih strojeva (engl. *Dalvik Virtual Machine*, DVM). Jezgrene knjižnice su knjižnice koje sadrže većinu jezgrenih knjižnica programskog jezika Java, a DVM pretvara Java klase u svoj vlastiti format .dex kako bi bile optimizirane za minimalni utrošak memorije.

Nakon knjižnica dolazi aplikacijski okvir koji se sastoji od mehanizma koji pomažu pisanje aplikacija, a na samom vrhu nalaze se sve aplikacije koje dolaze s uređajem, sve aplikacije koje se skidaju te sve aplikacije koje napišemo.

2.3. Programski jezik Java

Java je objektno-orijentirani programski jezik koji spada među najpopularnije jezike na svijetu. Baziran je na klasama, odnosno objektima, te je napravljen s ciljem da se sa što manje dodataka omogući razvoj aplikacija bilo koje vrste.

Prednost programskog jezika Java u odnosnu na većinu programskih jezika je što se programi pisani u Javi mogu izvoditi na svim operativnim sustavima za koje postoje Java virtualni stroj (engl. *Java Virtual Machine*, JVM). Odnosno, napisani kod je moguće pokrenuti na bilo kojem softveru (engl. *software*) i hardveru (engl. *hardware*) uz minimalne preduvjete, zahvaljujući bajtkodu (engl. *bytecode*).

Zbog velikog broja prednosti u odnosu na ostale programske jezike, Java je popularna za razvoj programa na mobilnim telefonima, a javlja se kao osnovni jezik za programiranje Android aplikacija. Prema slici 2.4. prikazan je jednostavan primjer programskog koda Java.

```
class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

Sl. 2.4. Primjer Java koda, prema [5].

2.4. Jezik za označavanje podataka (engl. *EXtensible Markup Language*, XML)

XML je dizajniran za pohranu i distribuciju podataka preko Interneta te je dizajniran na način da bude čitljiv i ljudima i strojevima. U izradi Android aplikacije XML se koristi kako bi se

definiralo korisničko sučelje. Pomoću XML-a postavljaju se gumbovi, polje za prikaz teksta, polje za unos teksta i mnoge druge komponente.

Prednost korištenja XML-a za izradu korisničkog sučelja je što se na taj način odvaja korisničko sučelje od koda koji kontrolira ponašanje aplikacije. Moguće je kreirati više korisničkih sučelja, za primjerice različite veličine ekrana, a da pri tome koristimo istu funkcionalnost aplikacije. XML se prilikom izrade Android aplikacije ne koristi samo kako bi definirao izgled korisničkog sučelja nego i za druge XML datoteke kao što su AndroidManifest.xml (deklariranje dijelova aplikacije koji će se koristiti u projektu te njihove predispozicije), strings.xml (sadrži sve vrijednosti niza znakova unutar aplikacije na jednom mjestu), colors.xml (isto kao i strings.xml samo za boje). Prema slici 2.5. vidljiv je primjer definiranja korisničkog sučelja na kojem se ispisuje "Hello world" u XML-u.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello world!" />

</LinearLayout>
```

Sl. 2.5. Primjer XML koda.

2.5. ZXing (Zebra Crossing)

Kako bi bilo moguće koristiti neke od funkcija koje Android daje na raspolaganje, potrebno je dodati biblioteku koja će omogućiti određenu funkciju, u ovom slučaju skeniranje bar koda ili QR koda. Potrebno je voditi računa koje biblioteke su dodane u određeni projekt te jesu li one dane na raspolaganje od pouzdanih osoba ili tvrtki. Vrlo je važno koristiti pouzdane biblioteke iz razloga što osobe koje su je kreirale imaju mogućnost njezine promjene ili brisanja, a to bi značilo da određeni projekt u kojem su bile prisutne, više ne radi na onaj način na koji se očekivalo da će raditi. Zebra Crossing biblioteka jedna je od onih koje su pouzdane i svakim danom se unapređuje. Kao što je spomenuto prije, koristi se za skeniranje bar koda i QR koda pomoću kamere na mobilnom uređaju te je vrlo jednostavna za korištenje.

2.6. Room

Room je biblioteka koja omogućava pohranu podataka jednako kao i SQLite, ali s puno više prednosti. Neke od prednosti su brže pohranjivanje, lakše korištenje te manje pisanje koda čime je smanjena mogućnosti pogreške kod pisanja SQL upita. Room je biblioteka napravljena od strane Google-a pa ju se zbog toga smatra pouzdanom.

2.7. Točka interesa Android (engl. *Point of interest Android*, POI)

Biblioteka POI je napravljena od strane Apache-a za manipuliranje raznih vrsta dokumenata, uključujući i .xls dokumente. Kako bi se biblioteka mogla koristiti na Android platformi, napravljena je POI Android biblioteka. U ovom radu bit će vrlo korisna jer je potrebno čitati podatke iz .xls dokumenta.

2.8. Bar kod

Ideja bar koda potiče iz SAD-a gdje je lokalni trgovački lanac inicirao razvoj sistema za učitavanje informacija o artiklu prilikom naplate na blagajni. Ovaj problem je zainteresirao dva inženjera koji su se u potpunosti posvetili ovom problemu i stvorili prvi bar kod pod nazivom bikovo oko (engl. *Bulls eye*). Zbog određenih nedostataka razvili su novi bar kod nazvan UPC (engl. *Universal Product Code*) koji je sadržavao 12 znamenki. Nakon UPC-a dolazi novi EAN kod koji sadržava 13 znamenki, prema [6]. Kako se bar kod počeo koristiti sve češće, samim tim počele su se razvijati i druge vrste kodova za različite primjene. Glavna podjela bar kodova je na jednodimenzionalne i dvodimenzionalne. Jednodimenzionalni bar kodovi u sebi nose samo jedan podatak, dok dvodimenzionalni nose niz informacija o samom proizvodu iz razloga što sadrže puno više znakova, čak 4000, u odnosu na jednodimenzionalne koji sadrže manje od 100. Među najpoznatije jednodimenzionalne spadaju: Codebar, Code 128, Code 39, EAN, JAN, UPC, a među dvodimenzionalne: Aztec, PDF417, DataMatrix, QR kod.

Bar kodovi (slika 2.6.) čitaju se sa posebnih skenerima. U trenutku kada je čitač usmjeren preko bar koda, tamne linije apsorbiraju svijetlost iz čitača, dok svijetli međuprostori reflektiraju svijetlost. Fotoćelija koja se nalazi u čitaču prima svijetlost i pretvara ju u električni signal koji se dekodira u čitaču u znakove koje bar kod predstavlja. Prve tri znamenke bar koda označavaju zemlju podrijetla, a zadnja znamenka označava kontrolni znak. Ostali brojevi između zemlje podrijetla i kontrolnog znaka označavaju proizvođača i proizvod.

Sl. 2.6. Bar kod, prema [7].

2.9. Podatkovna matrica (engl. *DataMatrix*)

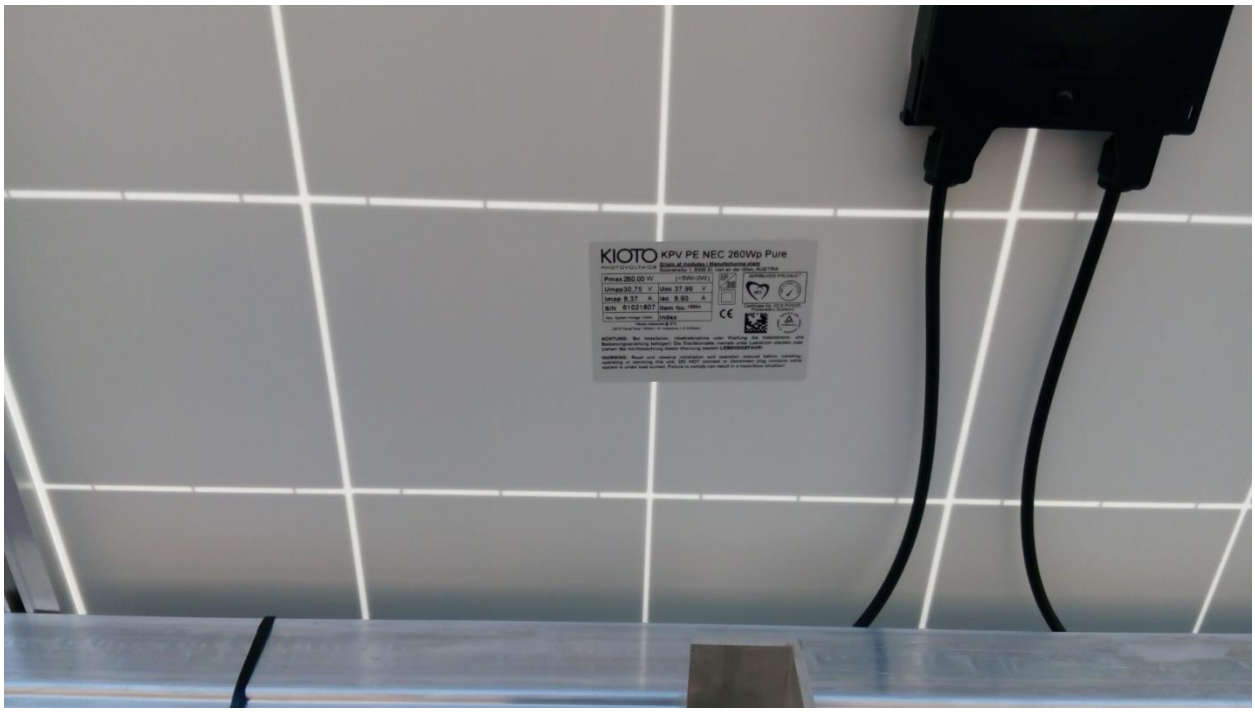
GS1 DataMatrix je dvodimenzionalna simbologija crtičnog koda. U literaturi se često koristi termin 2D crtični kod. DataMatrix kod može sadržavati do 2335 alfanumeričkih znakova i do 3116 numeričkih, prema [8]. Najčešće se primjenjuje u zdravstvu za izravno označavanje kirurških instrumenata. Postoje dvije vrste dvodimenzionalnih kodova:

- U obliku stoga
- U obliku matrice

Sadrže puno veći broj informacija nego jednodimenzionalni kodovi te na taj način mogu služiti kao sama baza podataka. Čitanje dvodimenzionalnih kodova zahtjeva čitače crtičnih kodova koji su skuplji i njihovo čitanje je sporije. Prednosti im je visoka količina podataka koju mogu sadržavati. Prema slici 2.7. vidljiv je primjer DataMatrix koda koji će biti korišten u ovom radu. Sastoji se od crnih i bijelih ćelija koje su raspoređene u obliku kvadrata ili pravokutnika. Ćelije predstavljaju bitove, a ovisno o potrebi kodiranja bijela ćelija predstavlja 0 dok crna ćelija predstavlja 1 ili obratno.

Sl. 2.7. DataMatrix kod, prema [9].

Svaki DataMatrix kod sastoji se od dviju susjednih granica u obliku slova „L“. Prva granica se naziva uzorak tražila (engl. *Finder pattern*) i služi za orijentaciju i lociranje simbola. Druga granica se naziva vremenski uzorak (engl. *Timing pattern*) i služi za oznaku broja redaka i stupaca u simbolu. Između dvije spomenute granice nalaze se redci i stupci informacija koji kodiraju informacije. Prema slici 2.8. prikazan je solarni panel na kojem se nalazi DataMatrix. kod.



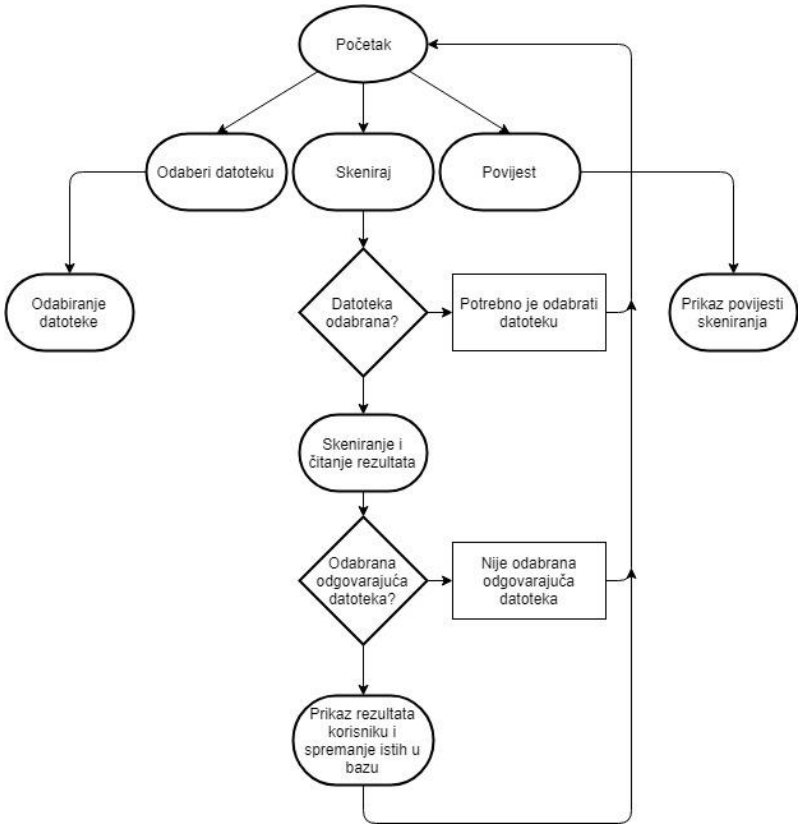
Sl. 2.8. Solarni panel na kojem se nalazi DataMatrix kod.

3. STRUKTURA APLIKACIJE

Aplikacija za optimalno raspoređivanje fotonaponskih panela sastoji se od nekoliko važnih dijelova kao i većina drugih Android aplikacija.

1. manifests – sadrži xml file AndroidManifest.xml u kojoj se definiraju dozvole koje aplikacija koristi, koji će *activity* biti prvi pokrenut i ostale važne informacije o aplikaciji
2. java – sve klase napisane u java programskom jeziku
3. res – sadrži 5 datoteka u kojima se nalaze resursi koji definiraju izgled aplikacije koji se prikazuje krajnjem korisniku:
    A. drawable – slike korištene u aplikacji
    B. layout – sadrži izgled svih *activity-a*
    C. menu – ako aplikacija ima dodatni menu ova datoteka sadrži xml file u kojemu se definira izgled menu-a
    D. minimap – izgled obične i okrugle ikonice za pokretanje aplikacije
    E. values – boje sadržane u aplikaciji, stilovi, tekst, itd.

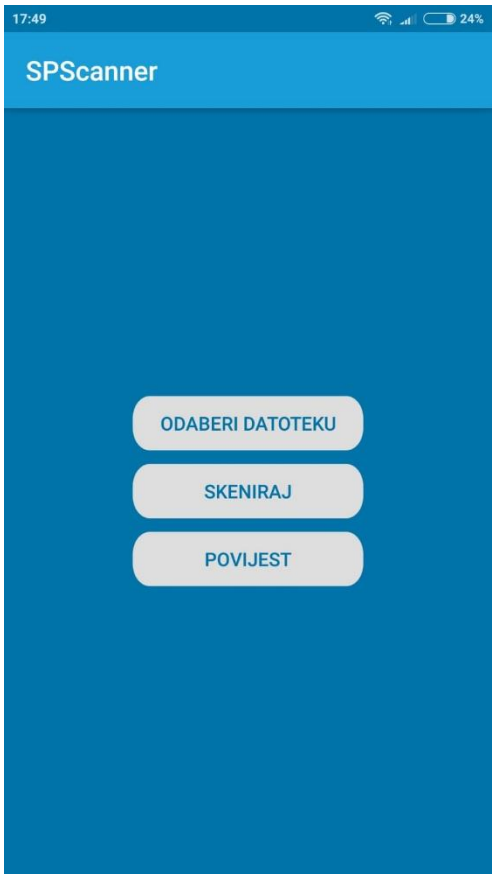
Kako bi snalaženje u aplikaciji bilo lakše snalazili u aplikaciji prikazan je dijagram tok aplikacije (Sl. 3.1.).



Sl. 3.1. Dijagram toka aplikacije.

3.1. Skeniranje koda

Prije ulaska u aplikaciju potrebno je dodati Excell dokument, koji sadržava informacije o solarnim panelima, na mobilni uređaj kako bi aplikacija bila u mogućnosti pročitati potrebne informacije. Prema slici 3.2. vidljivo je kako se ulaskom u aplikaciju otvara prozor koji sadrži tri gumba.



Sl. 3.2. Izgled početnog zaslona.

Prvi gumb služi za odabir Excell dokumenta, a druga dva gumba služe za otvaranje prozora za skeniranje i pregled povijest skeniranja. Izgled ovog prozora napisan je u .xml datoteci (Sl. 3.3.).

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:background="@color/colorAccent"
    android:gravity="center">

    <Button
        android:layout_width="170dp"
        android:layout_height="40dp"
        android:text="Odaberi datoteku"
        android:id="@+id/selectBtn"
        android:background="@drawable/button"
        android:textColor="@color/colorAccent"
        android:layout_marginBottom="10dp"/>

    <Button
        android:layout_width="170dp"
        android:layout_height="40dp"
        android:text="Skeniraj"
        android:id="@+id/scanBtn"
        android:background="@drawable/button"
        android:textColor="@color/colorAccent" />

    <Button
        android:layout_marginTop="10dp"
        android:layout_width="170dp"
        android:layout_height="40dp"
        android:text="Povijest"
        android:id="@+id/historyBtn"
        android:background="@drawable/button"
        android:textColor="@color/colorAccent"/>

</LinearLayout>
```

Sl. 3.3. Programski kod za definiranje izgleda početnog zaslona.

Klikom na gumb skeniraj pokreće se kamera na mobilnom uređaju sa crvenom linijom na sredini ekrana za orijentaciju. Moguće je da kod neće biti učitan istog trenutka pa je potrebno udaljiti kameru ili približiti kako bi se kod bolje vidio i u konačnici očitao.

Prema slici 3.5. vidljivo je korištenje Zebra Crossing biblioteke koja je spomenuta ranije.

```
IntentIntegrator intentIntegrator = new IntentIntegrator(activity);
intentIntegrator.setDesiredBarcodeFormats(IntentIntegrator.ALL_CODE_TYPES);
intentIntegrator.setPrompt("");
intentIntegrator.setCameraId(0);
intentIntegrator.setBeepEnabled(false);
intentIntegrator.initiateScan();
```

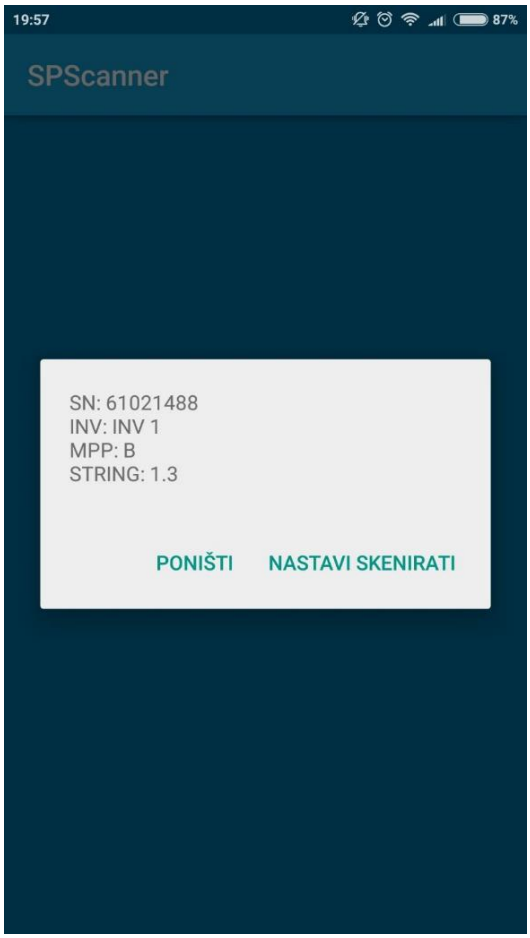
Sl. 3.4. Korištenje Zebra Crossing biblioteke.

Nakon što je kod skeniran, skenirani jedinstveni broj solarnog panela se prosljeđuje u klasu gdje se provjerava postoji li skenirani kod u Excell dokumentu (Sl. 3.5.). Kako bi bilo moguće pročitati Excell dokument potrebno je koristiti POI biblioteku. Prema slici 3.5. prikazan je dio koda pomoću kojeg se otvara Excell dokument kroz koji se prolazi sve dok se ne nađe skenirana vrijednost. Traženje skenirane vrijednosti obavlja se na način da se prolazi kroz svaki stupac dokumenta u kojem se nalaze vrijednosti svih kodova.

```
try {
            workbook = poiHandler.getWorkbook();
        } catch (FileNotFoundException e) {
            listener.unableToFindFile();
            return;
        }catch (IOException e){
            listener.unableToOpenFile();
            return;
        }


        for (int i = 7; i < 28; i++){
            cell = workbook.getSheetAt(0).getRow(i).getCell(3);
            if(scannedValue.equals(cell.toString())){
                inv = workbook.getSheetAt(0).getRow(3).getCell(3).toString();
                mpp = workbook.getSheetAt(0).getRow(1).getCell(9).toString();
                string = workbook.getSheetAt(0).getRow(5).getCell(4).toString();
            }
        }
```

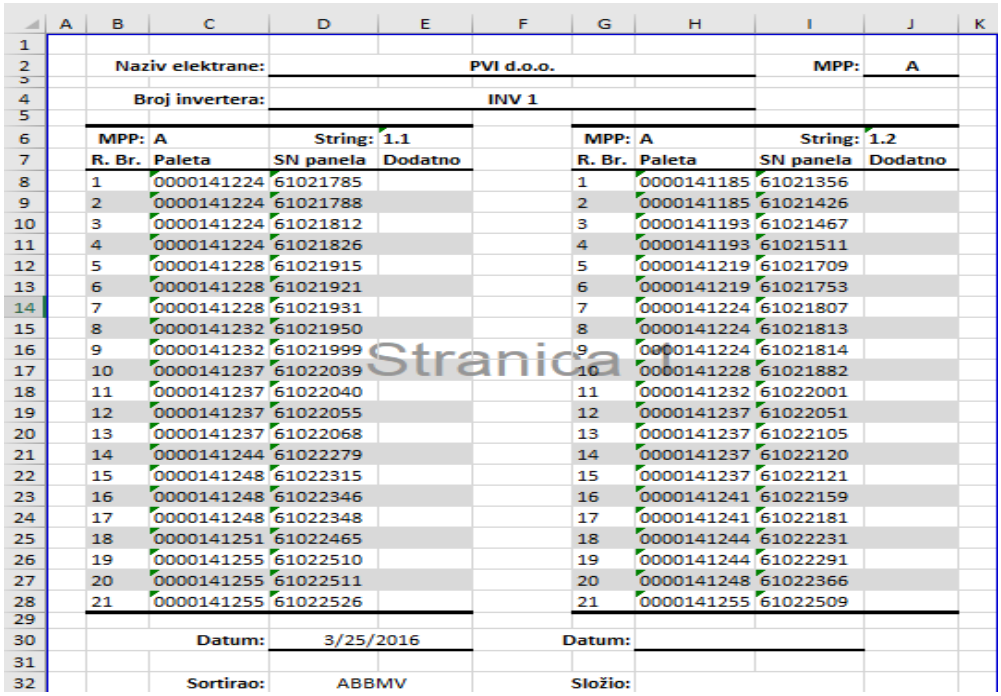
Sl. 3.5. Otvaranje Excell dokumenta i traženje skenirane vrijednosti.

Ako je vrijednost bar koda pronađena, dohvaćaju se sve njegove informacije koje stoje uz njega u dokumentu i šalju se u glavni prozor gdje se prikazuju u dijalogu upozorenja (Sl. 3.6.).



Sl. 3.6. Izgled dijaloga upozorenja.

Vrijednosti se dohvaćaju na osnovi stupca u kojem se nalazi skenirana vrijednosti. Ako se uzme za primjer da je skenirana vrijednost „61022068“ sa slike 3.7., u tom slučaju uzimaju se sve vrijednosti koje su dodijeljene za taj stupac (INV, MPP, String). Ako vrijednost bar koda nije pronađena u Excell dokumentu, ili ako dokument nije pronađen, korisnik će biti obaviješten.



		Naziv elektrane:		PVI d.o.o.				MPP:	A	
		Broj invertera:		INV 1						
		MPP:	A	String:	1.1		MPP:	A	String:	1.2
		R. Br.	Paleta	SN panela	Dodatno		R. Br.	Paleta	SN panela	Dodatno
		1	0000141224	61021785			1	0000141185	61021356	
		2	0000141224	61021788			2	0000141185	61021426	
		3	0000141224	61021812			3	0000141193	61021467	
		4	0000141224	61021826			4	0000141193	61021511	
		5	0000141228	61021915			5	0000141219	61021709	
		6	0000141228	61021921			6	0000141219	61021753	
		7	0000141228	61021931			7	0000141224	61021807	
		8	0000141232	61021950			8	0000141224	61021813	
		9	0000141232	61021999			9	0000141224	61021814	
		10	0000141237	61022039			10	0000141228	61021882	
		11	0000141237	61022040			11	0000141232	61022001	
		12	0000141237	61022055			12	0000141237	61022051	
		13	0000141237	61022068			13	0000141237	61022105	
		14	0000141244	61022279			14	0000141237	61022120	
		15	0000141248	61022315			15	0000141237	61022121	
		16	0000141248	61022346			16	0000141241	61022159	
		17	0000141248	61022348			17	0000141241	61022181	
		18	0000141251	61022465			18	0000141244	61022231	
		19	0000141255	61022510			19	0000141244	61022291	
		20	0000141255	61022511			20	0000141248	61022366	
		21	0000141255	61022526			21	0000141255	61022509	
			Datum:	3/25/2016			Datum:			
			Sortirao:	ABBMV			Složio:			

Sl. 3.7. Dio Excell dokumenta.

Na dijalogu upozorenja, osim što se nalaze informacije o skeniranom bar kodu, nalaze se i dva gumba pomoću kojih je korisniku dana mogućnost prestanka skeniranja ili nastavka skeniranja. Ako korisnik odabere nastavak skeniranja, isti postupak se ponavlja još jednom za isti ili različiti bar kod ovisno o izboru korisnika. Nakon što su informacije prikazane korisniku, one se spremaju u bazu pomoću Room biblioteke zajedno s datumom skeniranja i vrijednošću bar koda. Prema slikama 3.8. i 3.9. prikazan je programski kod za prikaz te za definiranje izgleda dijaloga upozorenja.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:padding="10dp"
        android:layout_margin="10dp"
        android:id="@+id/result_tv"/>

</LinearLayout>
```

Sl. 3.8. Programski kod za definiranje izgleda dijaloga upozorenja.

```
public void showResultsDialog(String barcode, String result) {

        presenter.storeResultsToDatabase(barcode, result);
        AlertDialog.Builder builder = new AlertDialog.Builder(this);
        View view = getLayoutInflater().inflate(R.layout.custom_dialog, null);
        TextView resultTv = view.findViewById(R.id.result_tv);
        resultTv.setText(result);

        builder.setNegativeButton("Poništi", new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialogInterface, int i) {
            }
        });
        builder.setPositiveButton("Nastavi skenirati", new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialogInterface, int i) {
                presenter.startScan();
            }
        });
        builder.setView(view);
        final AlertDialog dialog = builder.create();
        dialog.show();
    }
```

Sl. 3.9. Programski kod za prikaz dijaloga upozorenja.

3.2. Spremanje u bazu podataka

Kako bi bilo moguće spremiti podatke u bazu podataka, potrebno je kreirati odgovarajuće klase i sučelja koji su potrebni Room biblioteci za rad. Prva klasa koja se kreira je model koja daje uvid kako će tablica u bazi izgledati. Potrebno je dodati naziv tablice, id za svaki kreirani objekt i ostale podatke koji se spremaju. Podaci koji se spremaju u ovom slučaju su skenirani kod, vrijeme skeniranja i rezultat sa svim vrijednostima koje su pridružene skeniranom kodu (Sl. 3.11.).

```
@Entity(tableName = "scan_table")
public class Scan {

    @PrimaryKey
    @NonNull
    @ColumnInfo(name = "id")
    private String id;
    private String scannedValue;
    private String time;
    private String result;

    public Scan() {
    }

    public Scan(String scannedValue, String time, String result) {
        this.id = UUID.randomUUID().toString();
        this.scannedValue = scannedValue;
        this.time = time;
        this.result = result;
    }

    @NonNull
    public String getId() {
        return id;
    }

    public String getScannedValue() {
        return scannedValue;
    }

    public String getTime() {
        return time;
    }

    public String getResult() {
        return result;
    }

    public void setId(@NonNull String id) {
        this.id = id;
    }

    public void setScannedValue(String scannedValue) {
        this.scannedValue = scannedValue;
    }

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

    public void setResult(String result) {
        this.result = result;
    }
}
```

Sl. 3.11. Programski kod model klase.

Nakon što je uspješno kreiran model, kreira se *Data Access Object*, (DAO) klasa koja služi za manipuliranje podataka u bazi po pravilima koje sami definiramo. Prema slici 3.12. vidljiv je primjer korištenja DAO klase. Za potrebe ove aplikacije postoje tri pravila manipuliranja bazom:

1. Spremanje
2. Čitanje
3. Brisanje

```
@Dao
public interface ScanDao {

    @Insert
    void insert(Scan scan);

    @Query("SELECT * from scan_table")
    List<Scan> getAllResults();

    @Query("DELETE FROM scan_table")
    void delete();

}
```

Sl. 3.12. Kreiranje DAO sučelja.

Kako bi bilo moguće koristiti Room bazu podataka, ona zahtjeva kreiranje apstraktne klase koja sadrži osnovne informacije o bazi koja će se koristiti i njenu poveznicu kako bi se mogla koristiti kroz projekt. Neke od informacija koje je potrebno deklarirati su: korišteni entiteti u bazi, verzija baze, ime baze, itd. (Sl. 3.13). Prema slici 3.14. prikazano je spremanje rezultata u bazu podataka.

```
@Database(entities = {Scan.class}, version = 1)
public abstract class ScanRoomDatabase extends RoomDatabase {

    public abstract ScanDao scanDao();

    private static ScanRoomDatabase INSTANCE;

    public static ScanRoomDatabase getDatabase(final Context context) {
        if (INSTANCE == null) {
            synchronized (ScanRoomDatabase.class) {
                if (INSTANCE == null) {
                    INSTANCE = Room.databaseBuilder(context.getApplicationContext(),
                            ScanRoomDatabase.class, "scan_database")
                            .allowMainThreadQueries()
                            .build();
                }
            }
        }
        return INSTANCE;
    }
}
```

Sl. 3.13. Kreiranje Room baze podataka.

```
@Override
    public void storeResultsToDatabase(String scannedValue, String result) {
        Scan scan = new Scan(scannedValue, getTimestamp(), result);
        scanDao.insert(scan);
    }
```

Sl. 3.14. Spremanje rezultata u bazu podataka.

3.3. Pristup bazi podataka i prikaz rezultata korisniku

Čitanje podataka iz Room baze vrlo je jednostavno kao i ostale radnje nad samom bazom. Nakon što su podaci pročitani, potrebno ih je prikazati korisniku. U ovoj aplikaciji korišten je prikaz u obliku liste. Kako bi se prikazali rezultate u obliku liste, potrebno je kreirati dodatnu klasu koja služi za manipuliranje same liste (Sl. 3.15.) i izgled definiran u .xml datoteci za pojedini element liste (Sl. 3.16.).

```
public class ScanHistoryAdapter extends BaseAdapter {

    private List<Scan> scans;

    public ScanHistoryAdapter() {
        scans = new ArrayList<>();
    }

    public void updateScanResultsList(List<Scan> scan){
        scans.clear();
        scans.addAll(scan);
        notifyDataSetChanged();
    }

    @Override
    public int getCount() {
        return scans.size();
    }

    @Override
    public Object getItem(int position) {
        return scans.get(position);
    }

    @Override
    public long getItemId(int position) {
        return position;
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {

        TaskViewHolder holder = null;
        if(convertView == null){
            convertView = LayoutInflater.from(parent.getContext())
                .inflate(R.layout.item_scan, parent,false);

            holder = new TaskViewHolder(convertView);
            convertView.setTag(holder);
        }else{
            holder = (TaskViewHolder) convertView.getTag();
        }
        Scan scan = scans.get(position);

        holder.scannedValueTv.setText(scan.getScannedValue());
        holder.resultTv.setText(scan.getResult());
        holder.timestampTv.setText(scan.getTime());

        return convertView;
    }
```

Sl. 3.15. Programski kod klase za manipuliranje liste rezultata.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal"
    android:padding="5dp">

    <TextView
        android:id="@+id/num"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="0.4"
        android:gravity="left" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="0.2"
        android:id="@+id/value"
        android:gravity="center" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="0.4"
        android:id="@+id/timestamp"
        android:gravity="center"/>

</LinearLayout>
```

Sl. 3.16. Programski kod za definiranje izgleda pojedinog rezultata u listi.

Nakon što je klasa za manipuliranje listom uspješno napisana, ostaje još samo njeno pozivanje i prikazivanje podataka. Prema slici 3.17. vidljiva je klasa prozora povijesti gdje se obavlja pozivanje klase za manipuliranje listom i prosljeđivanje rezultata istoj. U slučaju da nema rezultata u bazi, korisniku će biti prikazana poruka da je povijest skeniranja prazna. U ovoj aplikaciji implementirana je i mogućnost brisanja povijesti skeniranja pritiskom na gumb koš za smeće koji se nalazi na alatnoj traci, isto kao i strelica koja korisnika vodi na prethodni, odnosno glavni prozor. Klikom na gumb koš za smeće brišu se svi rezultati iz baze podataka i postavlja se nova lista rezultata koja je prazna, što znači da će se prikazati poruka korisniku da je povijest skeniranja prazna.

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

    presenter = new HistoryPresenterImpl(App.getScanInteractor());
    presenter.setView(this);
}

@Override
protected void onResume() {
    super.onResume();
    presenter.readScansFromDatabase();
}


@Override
public boolean onCreateOptionsMenu(Menu menu) {
    super.onCreateOptionsMenu(menu);
    MenuInflater menuInflater = getMenuInflater();
    menuInflater.inflate(R.menu.custom_menu, menu);
    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch(item.getItemId()){
        case R.id.clear:
            presenter.deleteAllScansFromDatabase();
            presenter.readScansFromDatabase();
            break;
        default:
            finish();
    }
    super.onOptionsItemSelected(item);
    return true;
}

@Override
public void showNoScansTextView() {
    emptyHistoryTv.setVisibility(View.VISIBLE);
}

@Override
public void showScansListView(List<Scan> scans) {
    setupAdapter(scans);
}

private void setupAdapter(List<Scan> scans) {
    ScanHistoryAdapter adapter = new ScanHistoryAdapter();
    adapter.updateScanResultsList(scans);
    listView.setAdapter(adapter);
}
```

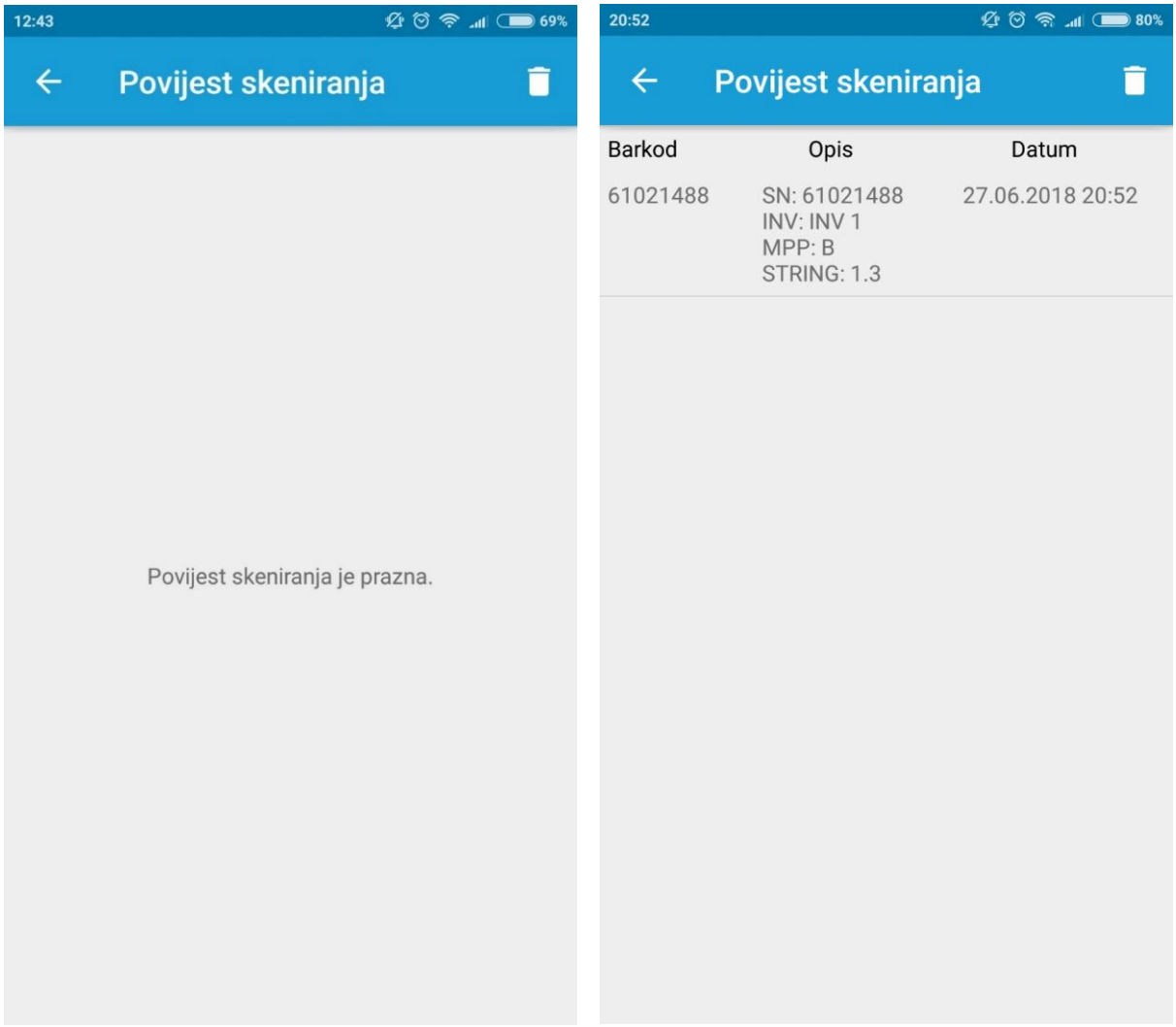
Sl. 3.17. Klasa prozora povijesti.

Prema slici 3.18. prikazan je kod za prikaz prozora povijesti skeniranja. On sadrži tri okvira za tekst koji formiraju zasebne stupce: Kod, Opis i Datum. Ispod njega nalazi se lista svih skeniranih rezultata koji su dohvaćeni iz baze podataka.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/linearLayout"
        android:orientation="horizontal"
        android:padding="5dp">

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="0.3"
            android:text="Barkod"
            android:textColor="#000"
            android:gravity="left"/>

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="0.16"
            android:text="Opis"
            android:textColor="#000"
            android:gravity="left"/>

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="0.4"
            android:text="Datum"
            android:textColor="#000"
            android:gravity="center"/>
    </LinearLayout>

    <ListView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/list_view"
        android:layout_below="@id/linearLayout"/>

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/emptyHistoryTv"
        android:layout_centerInParent="true"
        android:text="Povijest skeniranja je prazna"
        android:visibility="gone"/>

</RelativeLayout>
```

Sl. 3.18. Programski kod za definiranje izgleda zaslona povijesti

Slikom 3.19. prikazana su dva različita zaslona povijesti skeniranja. Na jednom zaslonu povijest je prazna, dok na drugom sadrži podatke jednog skeniranje.



Sl. 3.19. Izgled zaslona povijesti.

4. ZAKLJUČAK

Velika prisutnost mobilnih uređaja u svakodnevnom životu dovela je do korištenja istih u poslovne svrhe, a Android aplikacija za optimalno raspoređivanje fotonaponskih panela dobar je primjer toga.

U završnom radu napravljena je Android aplikacija pomoću koje skeniramo određeni bar kod. Ukoliko je vrijednost bar koda pronađena, dohvaćaju se i prikazuju sve informacije koje stoje uz njega u dokumentu, dok ako vrijednost bar koda nije pronađena korisnik će biti obaviješten o tome. Takvim postupkom rada informacije se pronalaze brže, jednostavnije, preglednije i lakše u odnosu na ručno traženje bar koda među dokumentima. Najveći naglasak je na uštedi vremena i olakšanju posla. Moguć je uvid u povijest svih dosadašnjih skeniranja, raspoređenih po datumu te mogućnost brisanja povijesti.

Aplikacija je vrlo jednostavna i praktična za korištenje i osim što sadrži sve funkcije koje su joj potrebne za učinkovitije i efikasnije obavljanje posla, odlična je podloga za veći projekt i dodatne nadogradnje, ukoliko su potrebne.

LITERATURA

[1] Skupina autora, Android (operacijski sustav) [online], Wikipedia, dostupno na: https://hr.wikipedia.org/wiki/Android_(operacijski_sustav), [28.06.2018.]

[2] Skupina autora, Android Gingerbread [online], Wikipedia, dostupno na: https://en.wikipedia.org/wiki/Android_Gingerbread, [28.06.2018.]

[3] Skupina autora, Android Pie [online], Wikipedia, dostupno na: https://en.wikipedia.org/wiki/Android_Pie, [28.06.2018.]

[4] Skupina autora, Android arhitektura [online], AndroidHub, dostupno na: http://www.androhub.com/android-architecture, [28.06.2018.]

[5] Skupina autora, Programski kod Java [online], Programiz, dostupno na: https://www.programiz.com/java-programming/hello-world, [28.06.2018.]

[6] Skupina autora, Osnovne informacije o bar kodovima [online], Wikipedia, dostupno na: https://en.wikipedia.org/wiki/Barcode, [28.06.2018.]

[7] Skupina autora, Bar kod [online], Burza,, dostupno na: https://burza.com.hr/portal/barkod-kao-izvor-informacija/5876, [28.06.2018.]

[8] Skupina autora, Informacije o DataMatrix kodu [online], GS1, dostupno na: https://www.gs1hr.org/hr/gs1-standardi/prikupljanje/gs1-datamatrix, [28.06.2018.]

[9] Skupina autora, DataMatrix [online], Wikipedia, dostupno na: https://en.wikipedia.org/wiki/Data_Matrix, [28.06.2018.]

SAŽETAK

Cilj ovog završnog rada bio je napraviti Android aplikaciju koja će omogućiti korisnicima efikasnije i učinkovitije dobivanje informacija o pojedinom fotonaponskom panelu putem skeniranja koda (Bar kod, QR kod) te na taj način olakšati rad. Korisnik ima mogućnost uvida u povijest skeniranja po datumu te mogućnost brisanja povijesti. Aplikacija je izrađena u Android Studio razvojnom okruženju uz pomoć Java programskog jezika i ostalih tehnologija. Aplikacija je vrlo jednostavna i praktična za korištenje. Kroz ovaj rad opisana je problematika i način na koji je riješena.

Ključne riječi: Android, aplikacija, baza podataka, Java, paneli

ABSTRACT

The goal of this paper is to make Android application which will allow users to more efficiently obtain information about a particular photovoltaic panel through scan code (Bar code, QR code) thus facilitate the work. The user is able to browse scan history by date and is able to erase history. The application was created in the Android Studio development environment with Java programming language and other technologies. The application is very easy and practical to use. Through this paper issues are described and the way in which they are solved.

Keywords: Android, applications, database, Java, panels

ŽIVOTOPIS

Antonio Butković rođen je 24. lipnja 1995. u Đakovu. 2002. godine započinje svoje školovanje u Osnovnoj školi Vladimir Nazor u Đakovu. Nakon završetka osnovne škole, 2010. godine upisuje Ekonomsku školu Braća Radić u Đakovu, smjer ekonomist. Nakon položene državne mature, 2014. godine upisuje Fakultet elektrotehnike, računarstva i informacijskih tehnologija u Osijeku, stručni studij elektrotehnike, smjer informatika.

_____

Antonio Butković

PRILOZI

CD

– Elektronička verzija rada (dokument u .docx i .pdf formatu)

– Android Studio projekt

– Aplikacija (.apk datoteka)