UNIVERZA NA PRIMORSKEM
FAKULTETA ZA MATEMATIKO, NARAVOSLOVJE IN
INFORMACIJSKE TEHNOLOGIJE

Zaključna naloga

Prototip mobilnega čitalca registrskih tablic za preverjanje parkirnin

(Prototype of mobile license plate reader for checking parking fee payments)

Ime in priimek: Klemen Lorenčič
Študijski program: Računalništvo in informatika
Mentor: izr. prof. dr. Tatjana Zrimec

Koper, september 2017

Ključna dokumentacijska informacija

Ime in PRIIMEK: Klemen LORENČIČ

Naslov zaključne naloge: Prototip mobilnega čitalca registrskih tablic za preverjanje parkirnin

Kraj: Koper

Leto: 2017

Število listov: 48 Število slik: 15 Število tabel: 1

Število prilog: 1 Število strani prilog: 1 Število referenc: 20

Mentor: izr. prof. dr. Tatjana Zrimec

Ključne besede: ALPR, čitalec registrskih tablic, Android, obdelava slike

Izvleček:
V zaključni nalogi je predstavljena izdelava prototipa mobilnega čitalca registrskih tablic. Po pregledu obstoječih rešitev in opisu motivacije za izdelavo aplikacije, sledi opis zasnove le-te. V poglavju namenjeno zasnovi, so opisane sistemske zahteve, vse ključne funkcije, ki so pričakovane v končnem produktu. Poleg tega je predstavljen še primer uporabe ter diagram pričakovanih aktivnosti. Sledi poglavje, kjer je natančno opisana implementacija prototipa. Opisani so tudi vsi koraki, ki se izvedejo pri prepoznavanju registrskih tablic. Ker gre za prototipno aplikacijo ima ta določene posebnosti, kot so dodajanje parkirišč, uporabnikov in transakcij, ter izpis rezultatov branja registrskih tablic. Po izdelavi prototipne aplikacije je bilo opravljeno tudi testiranje. V zaključni nalogi so predstavljene ugotovitve iz 100 vzorcev testiranja. Na podlagi ugotovitev, so bile opisane pomanjklivosti prototipa in možne izboljšave.

Key words documentation

Name and SURNAME: Klemen LORENČIČ

Title of final project paper: Prototype of mobile license plate reader for checking parking fee payments

Place: Koper

Year: 2017

Number of pages: 48 Number of figures: 15 Number of tables: 1

Number of appendices: 1 Number of appendix pages: 1 Number of references: 2

Mentor: Assoc. Prof. Tatjana Zrimec, PhD

Keywords: ALPR, license plate recognition, Android, image processing

Abstract:
The thesis presents stages of development of a prototype application for mobile license plate reader. Review of existing solutions and describing the motivation for developing the application is followed by a description of its design. In the design section, system requirements are described and all expected key functions in the final product. In addition, an example of use and a diagram of expected activities is presented. In the following chapter the implementation of the prototype is described in detail. All the steps taken to identify the license plates are also described. Because it is a prototype application, it has certain specific features such as adding parking lots, users and transactions, and displaying the results of recognized license plates. After the prototype application was made, testing was also performed. The final task presents findings from 100 test samples. On the basis of the findings, the shortcomings of the prototype and possible improvements were described.

Zahvala

Zahvalil bi se mentorci izr. prof. dr. Tatjani Zrimec za vso pomoč in odlično komunikacijo. Zahvalil bi se tudi celotni moji družini in prijateljem, ki so mi stali ob strani skozi celoten študij

Kazalo vsebine

1 Uvod **1**

2 Pregled obstoječih rešitev **2**
2.1 Prednosti . . . 2
2.2 Slabosti . . . 2
2.3 Motivacija za razvoj aplikacije . . . 3

3 Zasnova **4**
3.1 Opis aplikacije . . . 4
3.2 Sistemske zahteve . . . 4
3.3 Funkcije aplikacije . . . 4
3.3.1 Optično branje registrskih tablic . . . 4
3.3.2 Ročni vnos znakov registrske tablice . . . 4
3.3.3 Seznam parkirišč z lokacijo na zemljevidu . . . 4
3.3.4 Podrobnosti o registrski tablici . . . 5
3.3.5 Podpoglavje poglavja Široke vložitve . . . 5
3.4 Primer uporabe . . . 6
3.5 Diagram aktivnosti aplikacije . . . 7
3.6 Varnost . . . 8

4 Izvedba **9**
4.1 Uporabljene tehnologije . . . 9
4.1.1 Java . . . 9
4.1.2 Android Studio . . . 9
4.2 Zajem slike . . . 10
4.3 Obdelava slike . . . 10
4.3.1 Zaznavanje . . . 10
4.3.2 Binarizacija . . . 11
4.3.3 Analiza znakov . . . 11
4.3.4 Robovi tablice . . . 12
4.3.5 Poravnava . . . 12

4.3.6 Segmentacija znakov 12
4.3.7 OCR 13
4.3.8 Zaključna obdelava 13
4.4 Zemljevid 13
4.5 Posebnosti prototipa 14
4.6 Podatkovna baza 15
4.6.1 Realm 15
4.7 Uporabniški vmesnik 18

5 Testiranje **27**
5.1 Rezultati 27
5.2 Pomanjklivosti in možne izboljšave 28

6 Zaključek **29**

7 Literatura **30**

Kazalo tabel

1 Rezultati testiranja 27

Kazalo slik

1 Primer uporabe. 6
2 Diagram aktivnosti aplikacije. 7

3 Primer LBP algoritma. 11
4 Primer segmentacije znakov. 12
5 Primer registrske tablice s slovenskim vzorcem. 13
6 Shema podatkovne baze. 16
7 Zaslon za vpis. 18
8 Podzasloni glavnega zaslona. 19
9 Zaslon za izpis rezultatov in prižig kamere. 20
10 Uporabniški vmesnik Googlove kamerei. 21
11 Zaslon za ročni vnos. 22
12 Zaslon za prikaz informacij o tablici. 23
13 Zaslon za vnos uporabnikov. 24
14 Zaslon za vnos transakcij. 25
15 Zaslon za vnos parkirišč. 26

Kazalo prilog

A Izvorna koda
B Primeri testiranja

Seznam kratic

oz.	oziroma
SMS	sistem kratkih sporočil (angl. Short Message Servise)
URL	enolični krajevnik vira (angl. Uniform Resource Locator)
JSON	JavaScript Object Notation
XML	razširljivi označevalni jezik (angl. eXtensible Markup Language)
ALPR	samodejno zaznavanje registrske tablice (angl. Automatic License-Plate Recognition)
SDK	paket za razvoj programske opreme (angl. Software Development Kit)
LBP	lokalni binarni vzorci (angl. Local Binary Patterns)
OCR	optično prepoznavanje znakov (angl. Optical Characer Recognition)
API	vmesnik za namensko programiranje (angl. Application Programming Interface)

1 Uvod

Obremenjenost mest z vozili se iz leta v leto povečuje. Samo v prvih petih mesecih letošnjega leta 2017 se je registracija novih vozil v Evropski uniji povečala za 5,3% [1] in kot vse kaže se bo ta trend rasti nadaljeval. Občine vidijo to kot priložnost za zaslužek in v ta namen širijo obstoječa in gradijo nova plačljiva parkirišča. Vse več in vse večja parkirišča pa predstavljajo težavo za redarsko službo, predvsem v turistični sezoni, saj morajo le-ta pregledati večkrat dnevno. Do sedaj je bil pregled opravljen tako, da je redar moral najti listek v avtomobilu in pregledati, če je parkirnina potekla oz. do kdaj traja. Takšno pregledovanje pa je časovno potratno in v poletnih dneh tudi fizično zelo naporno. Vse več občin se tudi odloča za možnost plačila parkirnine preko SMS sporočila ali preko aplikacije za telefon, v tem primeru pa mora redar pregledati na drugi način ali je parkirnina plačana.

Cilj zaključne naloge je ustvariti aplikacijo, ki bo prepoznala znake registrske tablice ter izpisala vse podatke o transakciji parkirnine. S pomočjo kamere se zajameme slika, ki jo bo aplikacija predelala in vrnila znake registrske tablice, ki bodo nato uporabljeni za iskanje želenih podatkov.

Zaključna naloga je sestavljena iz šestih poglavij, vključno z Uvodom. Uvodu sledi poglavje, kjer so predstavljene obstoječe rešitve in motivacija za zaključno nalogo. V tretjem poglavju je predstavljena idejna zasnova sistema. Četrto poglavje vsebuje predstavitev implementacije ideje, kjer je predstavljen tudi sistema za prepoznavanje registrskih tablic. V petem poglavju so predstavljene ugotovitve testiranja aplikacije in pomankljivosti prototipa. Zadnje poglavje je zaključek, ki vsebuje povzetek celotne zaključne naloge.

2 Pregled obstoječih rešitev

Po iskanju potencialnih obstoječih rešitev sem prišel do ugotovitve, da trenutno na trgu ne obstaja oz. javno ni objavljena aplikacija za pametne naprave, ki služi pregledovanju parkirnine. Z nadaljnjo raziskavo, sem prišel do zaključka, da bi ta problem lahko bil rešen s pomočjo kamer oz. celostnih sistemov (kamera z računalnikom za obdelavo slik), ki se uporabljajo za nadzor prometa.

2.1 Prednosti

Glavna prednost takih sistemov, je predvsem hitrost in natančnost. Kamere teh vrst, lahko slikajo do 30 sličic na sekundo, računalniki, ki pa so povezani z njimi, pa imajo dovolj procesorske moči, da slike obdelajo v zanemarljivem času. Ker dobijo v obdelavo več enakih slik naenkrat, to poveča tudi natančnost rezultatov.

Take sisteme je možno tudi med sabo povezati, kar pomeni, da lahko še povečamo število možnih prepoznav tablic v želenem času.

2.2 Slabosti

Glavna slabost zgoraj omenjene rešitve je cena. Taki sistemi so praviloma zelo dragi, saj naročnik plačuje tako strojno opremo kot programsko opremo za obdelavo slik. Ker gre za zaprte sisteme in kakršen koli poseg v njih, za prilagoditve željam uporabnikov ni mogoč, bi bilo potrebno razviti še aplikacijo, ki bi komuniciral z omenjenim sistemom in z podatkovno bazo, kjer se iščejo želeni podatki. To še dodatno poveča že tako velike stroške.

Druga slabost take rešitve je izpostavljenost kamer. Kamere bi bile nameščene na vozila, kar pomeni, da so izpostavljene različnim elementom okolja in kljub zaščiti bi lahko prišlo do poškodb oz. okvar.

2.3 Motivacija za razvoj aplikacije

Motivacijo za zaključno nalogo sem črpal iz dejstva, da bi lahko z aplikacijo, ki sem si jo zamislil drastično zmanjšal strošek uporabnikom, ki potrebujejo tako rešitev. Moja rešitev, bi potrebovala, le pametno napravo, saj iz leta v leto se kamere na pametnih napravah izboljšujejo, ki so tudi procesorsko vse bolj močnejše.

3 Zasnova

3.1 Opis aplikacije

Sistem bo izdelan za redarske službe, ki bi rade svojim delavcem omogočile lažje in hitrejše pregledovanje parkiranih vozil. Uporabnik bo lahko s pomočjo aplikacije za Android pametne naprave ugotovil ali ima uporabnik parkirišča plačano parkirnino, do kdaj traja oz. kdaj je potekla.

3.2 Sistemske zahteve

Celoten sistem temelji na odprtokodnih rešitvah. Aplikacija za pametne naprave bo implementirana v programskem jeziku Java, s pomočjo knjižnice OpenALPR. Sistem bo lahko dopolnjen s strežniško aplikacijo ter podatkovno bazo že obstoječega ponudnika rešitve za parkiranje preko aplikacije ali kratkega sporočila. Lokalno bodo podatki shranjeni v podatkovni bazi Realm.

3.3 Funkcije aplikacije

3.3.1 Optično branje registrskih tablic

Branje registrskih tablic je glavna funkcija tega sistema. Optično branje je razdeljeno na dva dela, zajem slike ter na obdelavo le-te. Slika bo zajeta s pomočjo Google-ove kamere in bo posredovana v obdelavo OpenALPR knjižnici.

3.3.2 Ročni vnos znakov registrske tablice

Uporabniku bo poleg optičnega branja tablice omogočen tudi ročni vnos, saj lahko slabe vremenske in svetlobne razmere onemogočijo dovolj kvaliteten zajem slike za obdelavo.

3.3.3 Seznam parkirišč z lokacijo na zemljevidu

Uporabniku bo na voljo seznam vseh parkirišč, nad katerimi ima pristojnost. Ta bodo priročno tudi prikazana na zemljevidu.

3.3.4 Podrobnosti o registrski tablici

Uporabniku bodo prikazane vse ključne informacije o parkirnini za iskano tablico. Redar bo tako vedel za katero parkirno cono je bila parkirnina plačana, kdaj je bila plačana in kdaj poteče oz. je potekla.

3.3.5 Podpoglavje poglavja Široke vložitve

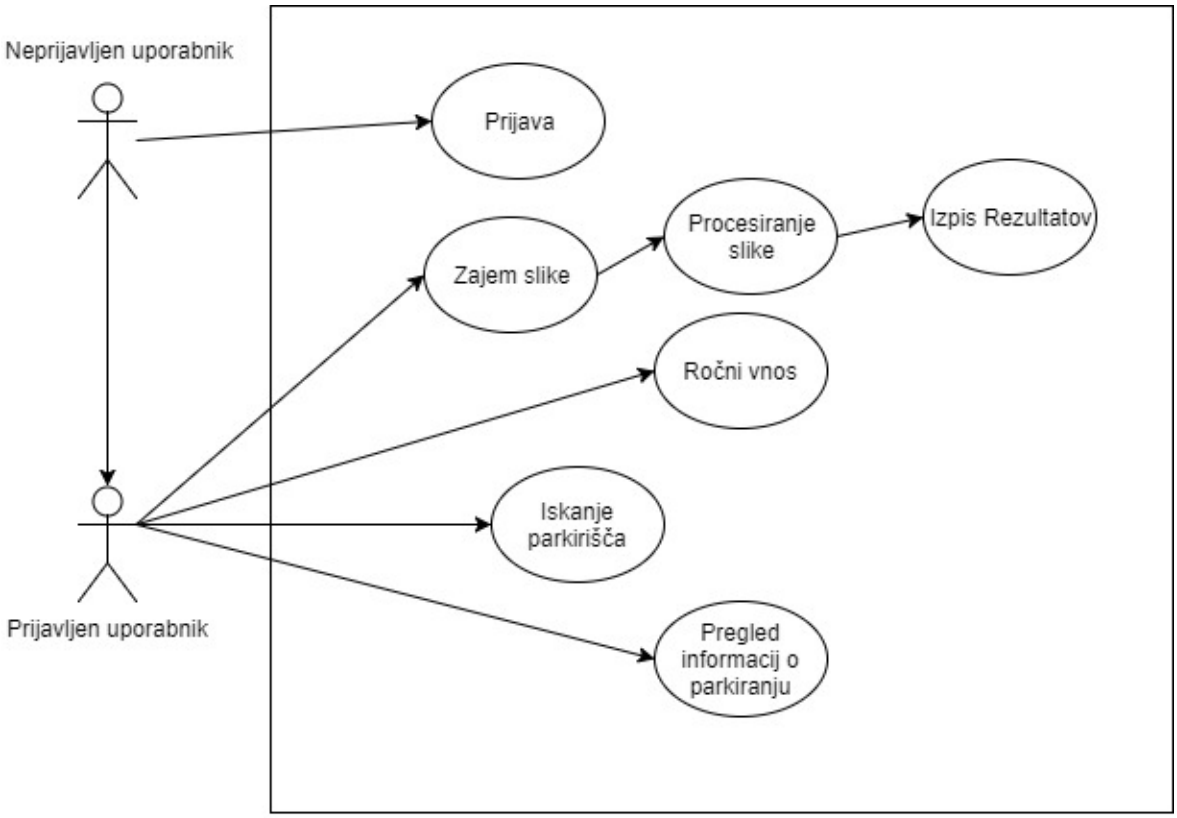
Aplikacija bo komunicirala preko URL naslova s strežnikom ponudnika rešitve za parkiranje preko aplikacije ali kratkega sporočila. Za izmenjavo podatkov bo uporabljen format JSON, z GET in POST zahtevo. Ta tekstovni format je idealen jezik za izmenjavo podatkov, saj je v celoti neodvisen od programskega jezika. [3]
JSON temelji na dveh strukturah:

- Zbirka parov ime/vrednost. V različnih jezikih je realizirana kot objekt, zapis, struktura, slovar, razpršena tabela ali asociativno polje.
- Urejen seznam vrednosti. V večini jezikov je realiziran kot polje, vektor, seznam ali zaporedje. [4]

V primerjavi z drugim prav tako zelo uporabljenim formatom za izmenjavo podatkov XML je JSON lažje berljiv in bolj pregleden, saj ni tako razmetan s značkami kot XML.

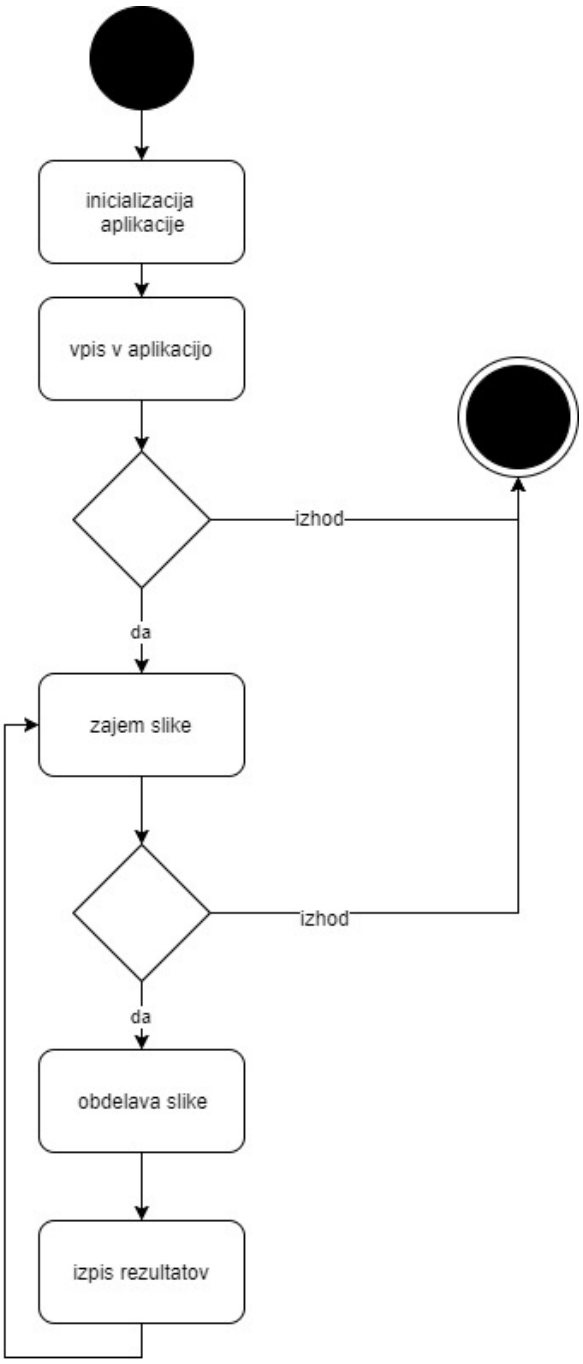
3.4 Primer uporabe

Uporabnik se bo prijavil v aplikacijo, kjer bo lahko izbiral med iskanjem registrske tablice ali pregledom seznama parkirišč. V zavihku za iskanje tablice bo se nato odločal med optičnim iskanjem. V primeru, da pogoji to ne dovoljujejo, bo pa lahko ročno vnesel znake tablice. Slika oz. znaki se nato pošljejo v nadaljnjo obdelavo.



Slika 1: Primer uporabe.

3.5 Diagram aktivnosti aplikacije



Slika 2: Diagram aktivnosti aplikacije.

Potek izvanjanja programa:

1. Uporabnik zažene aplikacijo in ta se inicializira
2. Uporabnik vnese uporabniško ime in geslo
   (a) Alternativni potek: uporabnik zapusti aplikacijo
3. Uporabnik zažene kamero in zajame sliko
   (a) Alternativni potek: uporabnik zapusti aplikacijo
4. Aplikacijo obdela sliko
5. Aplikacije prikaže podatke na podlagi obdelane slike
6. Vrnemo se na stanje 3

3.6 Varnost

Ker gre pri registrskih tablicah za zelo občutljive podatke, je potrebno za varnost dobro poskrbeti. Uporabnik bo imel dostop le do tablic na parkiriščih na katerih ima pristojnost. Za dodatno varnost bo še poskrbljeno tako, da ne bo možno narediti sliko zaslona pametne naprave.

4 Izvedba

4.1 Uporabljene tehnologije

Fazo izvedbe sem se lotil z izbiro primernih orodji in okolji, ki so mi poenostavili razvoj in izvedbo mojega sistema. Android omogoča razvoj aplikacij v kopici jezikov, med najbolj priljubljenimi jeziki so Java, HTML5, C# (Xamarin) in Corona, po novem tudi Kotlin.

Za programske jezike sem si izbral Javo. Java je bila uporabljena za razvoj Android aplikacije v razvojen okolju Android studio.

4.1.1 Java

Java je splošno namenski objektno orientiran programski jezik. Je zasnovan dovolj preprosto, da lahko programerji dosežejo tekočnost v jeziku. [2] Za Javo sem se odločil, zaradi predhodnega znanja ter obsežne dokumentacije. Poleg tega je na internetu tudi močna skupnost Java Android programerjev, na katere se lahko obrneš v primeru težav.

Za začetek razvoja sem moral najprej namestiti Javo in Java razvojno orodje (ang. Java Development Kit), nato sem nadaljeval s namestitvijo Android Studi-a.

4.1.2 Android Studio

Android Studio je uradno integrirano razvojno orodje za Android aplikacije. Je namensko narejeno za pospešitev in pomoč pri razvoju le-teh. Za to razvojno okolje sem se odločil, saj v njem lahko razvijamo tako funkcionalnosti kot uporabniški vmesnik aplikacije, poleg tega pa je še preprosto testirati razvite aplikacije, saj le priklopimo svojo Android napravo na računalnik ter preko Android Studia na njej tudi zaženemo aplikacijo. V primeru, da si ne lastimo Android naprave lahko izberemo za testiranje enega od prednaloženih simulatorjev. [5, 6]

Med namestitvijo Android Studija sem namestil tudi Android SDK knjižnico. Ta knjižnica vsebuje vsa potrebna orodja za razvoj želene aplikacije.

4.2 Zajem slike

Zajem slik bo izveden s pomočjo Googlove kamere. Prednost Googlove kamere je, da ni potrebno zahtevati pravico za uporabo, saj praviloma uporabniki potrdijo dovoljenje za uporabo kamere že ob prvi uporabi telefona. Med samim zagonom kamere aplikacija prosi uporabnika še za dovoljenje dostopanja in upravljanja zunanje pomnilniške shrambe. Zahtevane uporabniške pravice, jih je potrebno specificirati v datoteki "AndroidManifest" na sledeči način:

```
<uses-permission android:name=
"android.permission.WRITE_EXTERNAL_STORAGE" />
```

Dostop do notranje shrambe potrebujemo, saj shranjujemo zajete slike, ki služijo za nadaljnjo obdelavo. Vsaka slika je shranjena s unikatnim imenom, zgrajenim iz datuma in ure ("yyyy-MM-dd-hh-mm-ss") v jpg formatu. Shranjena slika je nato poslana v nadaljnjo obdelavo v OpenALPR knjižnico.

4.3 Obdelava slike

Obdelavo slik izvede OpenALPR. OpenALPR je odprtokodna knjižnica za samodejno prepoznavanje registerskih tablic napisana v C++ z oprimki v Javi, C# ter Python-u. Knjižnico je bilo potrebno integrirati v projekt ter jo pravilno inicializirati. V inicializaciji je potrebno podati razred, v katerem bo knjižnica zagnana, pot do datoteke, kjer je shranjena slika ter katere vzorce tablic se pričakuje in koliko možnih pravilnih rezultatov želimo dobiti nazaj. Primer inicializacije lahko vidimo v spodnji kodi:

```
OpenALPR.Factory.create(ScannActivity.this, ANDROID_DATA_DIR)
.recognizeWithCountryRegionNConfig("eu", "",
destination.getAbsolutePath(), openAlprConfFile, 10);
```

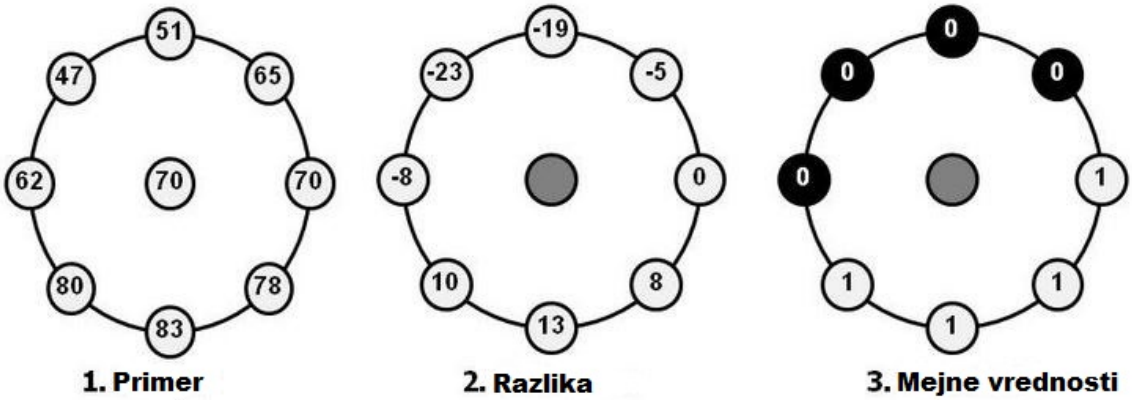
Obdelava slik deluje po principu cevovoda. Vhod je slika, različne operacije se izvedejo v fazah in kot izhod dobimo možne znake registrske tablice na sliki. Faze cevovoda so predstavljene v naslednjih podpoglavjih. [7]

4.3.1 Zaznavanje

Faza zaznavanja se izvede enkrat za vsako vhodno sliko. Za zaznavo območja tablice je uporabljen LBP algoritem.

Osnovna ideja algoritma za lokalne binarne vzorce je povzeti lokalno strukturo na sliki, z primerjanem vsakega piksla z okolico. Vzame se osrednji piksel ter se odštej

njegav prag od sosednjih, če je intenzivnost sredinskega piksla večja ali enaka od sosednjega, se ga označi z 1 v nasprotnem primeru pa z 0. Po zaključku dobimo binarno število piksla, na primer 1111000 (Slika 3). Z osmimi sosednjimi piksli dobimo 2^8 kombinacij, ki jih imenujemo lokalni binarni vzorec. [8]



Slika 3: Primer LBP algoritma.

4.3.2 Binarizacija

Ta, ter vse naslednje faze se izvedejo večkrat, enkrat za vsako možno prepoznavo registrske tablice.

Faza binarizacije bo ustvarila več možnih binarnih slik za vsako območje na registrski tablici. Potrebujemo več slik, saj nam to zagotavlja boljše možnosti za prepoznavo znakov na registrski tablici. Če bi imeli samo eno sliko, bi ta lahko bila presvetla ali pretemna in znak nebi bil prepoznan. Binarizacija uporablja Wolf-Jolien in Sauovola metodo. [9]

Ti metodi sta uborabljeni za razčlenjevanje slike. Metode za razčlenjevanje delujejo tako, da vsak piksel na sliki zamenjajo z črnim, če je inteziteta območja I_i, j slike manjša od določene konstante, v nasprotnem primeru pa se piksle zamenja za bele. [10]

4.3.3 Analiza znakov

Analiza znakov bo poskušala najti regije na registrski tablici v velikosti črk. Ta faza je izvedena v dveh korakih. Prvi korak poišče vse povezane "packe" na območju registrske tablice. V drugem koraku so poiskane "packe" ki ustrezajo višini in širini znakov registrske tablice ter so vrh in dno poravnani med ostalimi podobnimi "packami". Analiza bo opravljena večkrat in bo iskala znake po velikosti od najmanjšega do največjega.

V primeru, da v tem koraku ni najdeno nič, bo operacija obdelave slike prekinjena, v nasprotnem primeru pa se bo zaznana regija shranila in bila poslana v nadaljnjo obdelavo. [11]

4.3.4 Robovi tablice

V tej fazi bodo najdeni robovi registrske tablice. To je potrebno, saj v fazi detekcije je najdeno le potencialno območje registrske tablice, ki je praviloma malo manjše ali večje, v tej fazi pa bo bilo določeno dejansko območje.

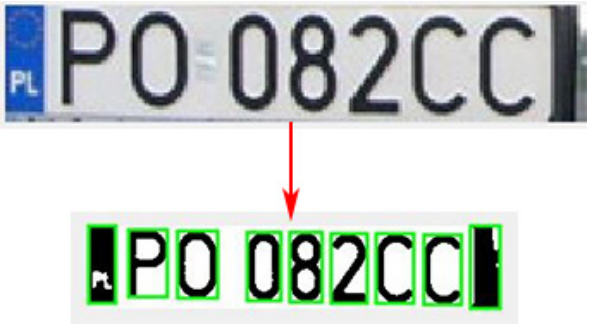
V prvem koraku so poiskane vse povezane črte, slika registrske tablice bo obdelana in sestavljen bo seznam vseh navpičnih in vodoravnih črt. Seznam črt in višina znakov bodo uporabljeni za določiti robove registrske tablice, ki bodo nato primerjani z optimalnimi robovi. [12]

4.3.5 Poravnava

Sledi faza poravnave. Ta faza poravna območje registrske tablice na standardno velikost in orientacijo. Ideja te faze je, da pridobimo praviloma orientirano sliko registrske tablice brez popačitve. [13]

4.3.6 Segmentacija znakov

V tej fazi se poskuša osamiti vse znake na registrski tablici. Uporabljen bo navpični histogram za iskanje praznin v znakih registrske tablice. Prav tako se v tej fazi odstrani nepovezane packe in regije znakov, ki so prekratke. Odstranilo bo se tudi robe tablice, da nebi bili zaznani kot "I" ali "1". [14]



Slika 4: Primer segmentacije znakov.

4.3.7 OCR

OCR faza bo analizirala vsak znak posamično. Za vsako sliko znaka bo določen možen znak. [15]

Obstajata dva osnovna algoritma, ki vrnejo seznam možnih znakov. Primerjanje matrik, primerja vsak piksel na sliki, s pikslom shranjene slike znaka. Ta način se zanaša na to, da je znak iz slike pravilno izoliran, ter da se ujema s fontom shranjenega znaka. Zaradi spreminjanja fontov ta algoritem ni najbolj zanesljiv.

Drugi algoritem je dvofazni način. Tak algoritem uporablja programska oprema Tesseract, ki je tudi uporabljena za to fazo. V prvi fazi so prepoznani znaki z visokim procentom sigurnosti, ki so nato uporabljeni v drugi fazi kot podlaga pri pomoči prepoznave še ne prepoznanih znakov. [16]

4.3.8 Zaključna obdelava

Ko faza OCR vrne vse prepoznane znake, se v zaključni fazi sestavijo najboljše možne kombinacije znakov. Ta faza vrne najboljših “n” (koliko možnih rešitev želimo, smo definirali v inicializaciji) kombinacij znakov. Zaključna obdelava bo izločila znake pod določeno mejo procenta sigurnosti, ter jih bo nadomestila z praznim znakom.

Zaključna obdelava bo tudi pregledala ali ustreza vzorcu, navedenem v inicializaciji (npr. slovenski vzorec: [črka][črka]-[številka][številka][številka]), ter na podlagi tega izbrala najboljše kombinacije. [17]

Slika 5: Primer registrske tablice s slovenskim vzorcem.

4.4 Zemljevid

Zemljevidu je posvečen celoten Fragment, zaradi boljše preglednosti. Da lahko mapo sploh uporabimo, je potrebno projekt prijavit pri Google-u, kjer dobimo API ključ, ki omogoči uporabo mape. [18] Osnovni ključ omogoča 250000 dnevnih dostopov do mape. [19] Za aktivacijo zemljevida, je pridobljen API ključ potrebno vnesti v AndroidManifest na sledeči način:

```
<meta-data
    android:name="com.google.android.geo.API_KEY"
    android:value="AIzaSyD1X0e8SZLlSl7FnwYdsZus28R5XthIsww"/>
```

Ob vsakem zagonu aplikacije je mapo potrebno inicializirati, kjer tudi določimo na katerem zaslonu naj bo vidna:

```
googleMap = ((SupportMapFragment) getChildFragmentManager()
.findFragmentById(R.id.map_fragment)).getMap();
...
MapsInitializer.initialize(getActivity()
.getApplicationContext());
```

V primeru, da imamo v bazi shranjena parkirišča, jih označimo na zemljevidu s rdečimi značkami. Znački parkirišča določimo lokacijo na podlagi decimalnih vrednosti geografske dolžine in širine, ki so vnešene v bazo:

```
Marker marker = googleMap.addMarker(new MarkerOptions()
.position(new LatLng(pp.getLatitude(), pp.getLongitude()))
.title("Cona: " + pp.getPPKeyword() + " - " + pp.getPPName()));
```

Značkam lahko določimo tudi vsebino. S pritiskom na značko se bo izpisalo celotno ime parkirišča ter njegova ključna beseda.

4.5 Posebnosti prototipa

Ker gre za prototipno aplikacijo, je bilo določene stvari potrebno simulirati. Aplikacija ni integrirana z nobenim sistemom od obstoječih ponudnikov elektronskega plačevanja parkirišč, zato je podatke za testiranje potrebno pridobiti drugače. Ker nisem želel imeti statičnih podatkov pri testiranju, sem ustvaril tri aktivnosti za dodajanje podatkov v bazo in sicer za dodajanje parkirišč, dodajanje parkirnih transakcij ter dodajanje uporabnikov. Do vseh treh aktivnosti dostopamo preko t.i. menija s tremi pikicami (ang. 3 dot menu):

- **Dodajanje parkirišč:**
  Ko dodajamo parkirišča je potrebno vnesti šifro parkirišča (npr. KP1, LJ12, ...), ime parkirišča, ter njegovo geografsko širino in dolžino.

- **Dodajanje parkirnih transakcij:**
  Ta aktivnost simulira podatke, ki so pridobljeni, ko uporabnik plača parkiranje

preko tekstovnega sporočila ali preko aplikacije, če ta obstaja. Ključni podatki transakcije, ki bi bili uporabni za redarje so šifra parkirišča, registrska tablica, ter čas in datum začetka parkiranja in kdaj parkirnina poteče (oboje v obliki yyyy-MM-dd hh:mm:ss)

- **Dodajanje uporabnikov:**
  Za namen prototipa se za uporabnika doda samo uporabniško ime in geslo.

4.6 Podatkovna baza

V aplikacijo je implementirana tudi podatkovna baza. V prototipu služi, za shranjevanje podatkov iz prejšnjega poglavja. Bazo aplikacije sestavljajo tri sheme (Slika 6):

- **Wardens:**
  Ta shema je namenjena shranjevanju uporabniških imen in gesel od uporabnikov. Ta shema služi le v namene testiranja, saj iz varnostnega vidika, to ni primeren način shranjevanja takega tipa podatkov.

- **ParkingPlaces:**
  V tej shemi bodo shranjeni vsi podatki, ki se navezujejo na parkirišče, torej ključna beseda, ime ter geografska dolžina in širina.

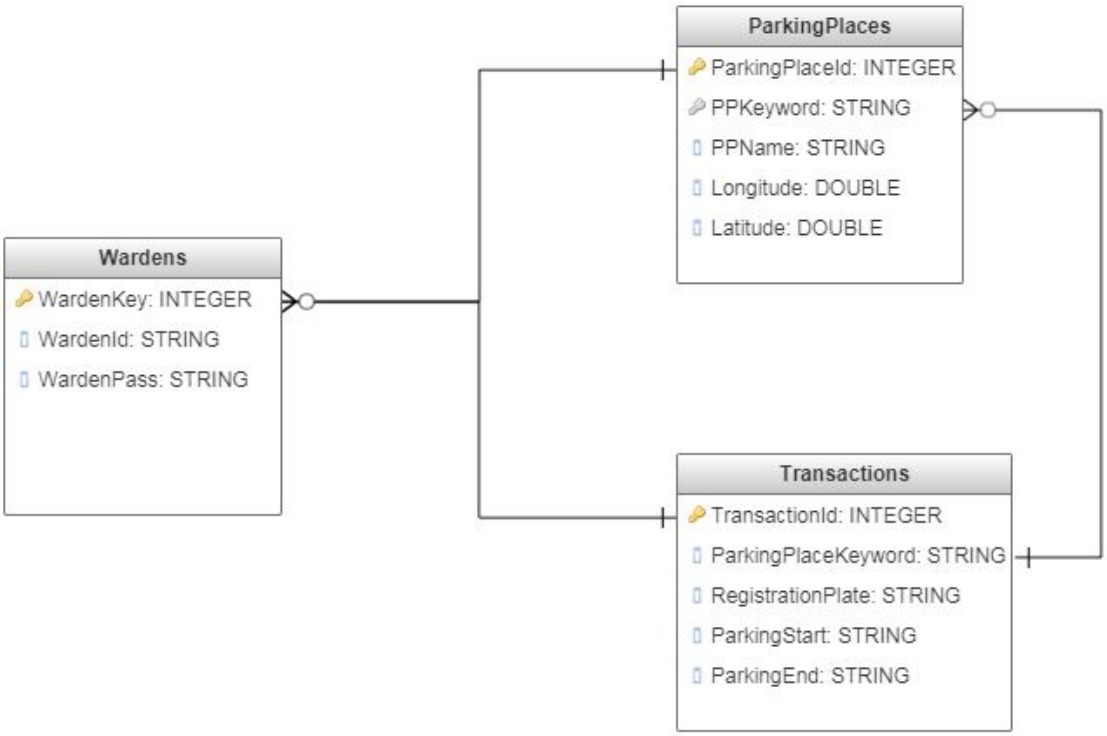
- **Transactions:**
  Ta shema je namenjena shranjevanju ključnih podatkov transakcij plačanih parkirnin. Vsebuje ključno besedo parkirišča, registrsko številko za katero je bila plačana parkirnina, ter čas pričetka in kdaj parkrinina poteče.

Za implementacijo podatkovne baze v aplikacijo, sem uporabil rešitve Realm-a.

4.6.1 Realm

Osnovni koncept Realm-a je lahki vsebnik predmetov. Tako kot v podatkovni bazi, tudi v Realm lahko iščemo elemente, jih povezujemo in filtriramo. V nasprotju s tradicionalno podatkovno bazo, so objekti v Realm.u "živi" in odzivni. [20] V prvem koraku, je potrebno ustvariti razred, ki je Realm objekt:

```
public class Wardens extends RealmObject{
...
}
```



Slika 6: Shema podatkovne baze.

V tem razredu nato ustvarimo želene elemente, ter t.i. "get-erje" in "set-erje" s pomočjo katerih bomo zapisovali v Realm in pridobivali podatke iz njega.

```
private String WardenId;
public String getWardenId() {return WardenId;}
public void setWardenId(String wardenId) {
   WardenId = wardenId;}
```

Ko imamo ustvarjen Realm objekt lahko začnemo dodajati elemente v le-tega. To storimo tako, da ustvarimo nov primer Realma in zaženemo novo Realm transakcijo, nato pa s pomočjo set-erjev dodelimo vrednosti. Na koncu pa to Realm transakcijo izročimo:

```
Realm pWarden = Realm.getInstance(realmConfiguration);
pWarden.beginTransaction();
parkingWarden.setWardenId(wrUsername.getText().toString());
pWarden.commitTransaction();
```

Iskanje po Realm-u se prav tako začne z ustvarjanjem primera Realma in zagonom nove Realm transakcije. Tako kot je že bilo zgoraj omenjeno, nam Realm omogoče

filtriranje, kar pomeni, da lahko iz baze izluščimo le podatke, ki nas zanimajo. Na primer vse podatke, ki vsebujejo določeno registrsko številko:

```
transRegPlate = regPlates.where(Transactions.class)
.equalTo("RegistrationPlate", regPlate).findAll();
```

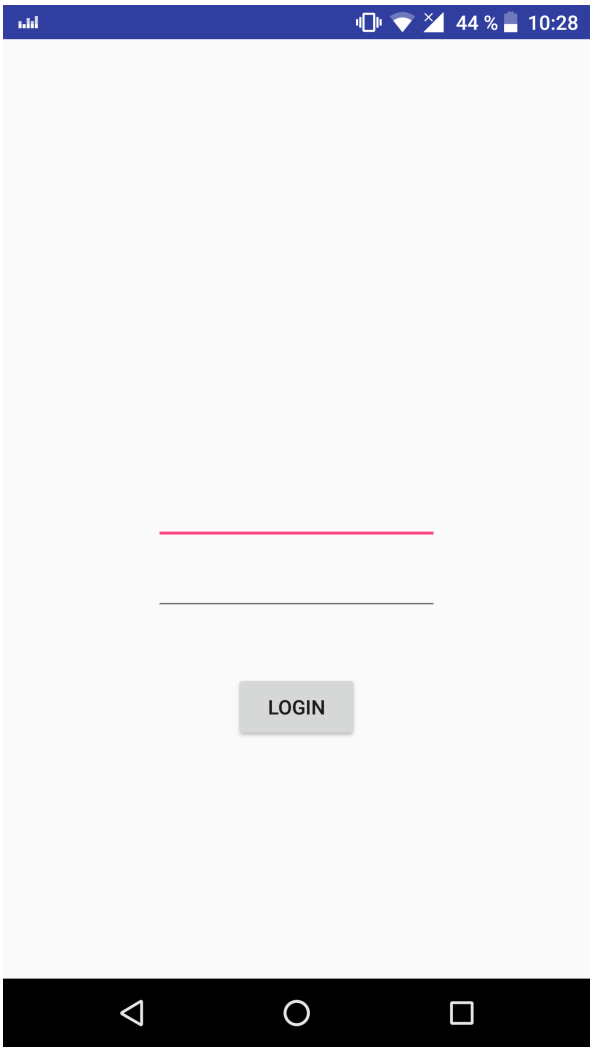
Spremenljivka "transRegPlate" je tipa Realm, ki vsebuje elemente tipa Transactions, kar pomeni, da lahko iz nje pridobivamo podatke s pomočjo predhodno ustvarjenih get-erjev:

```
transRegPlate.getParkingPlaceKeyword();
transRegPlate.getRegistrationPlate();
```

4.7 Uporabniški vmesnik

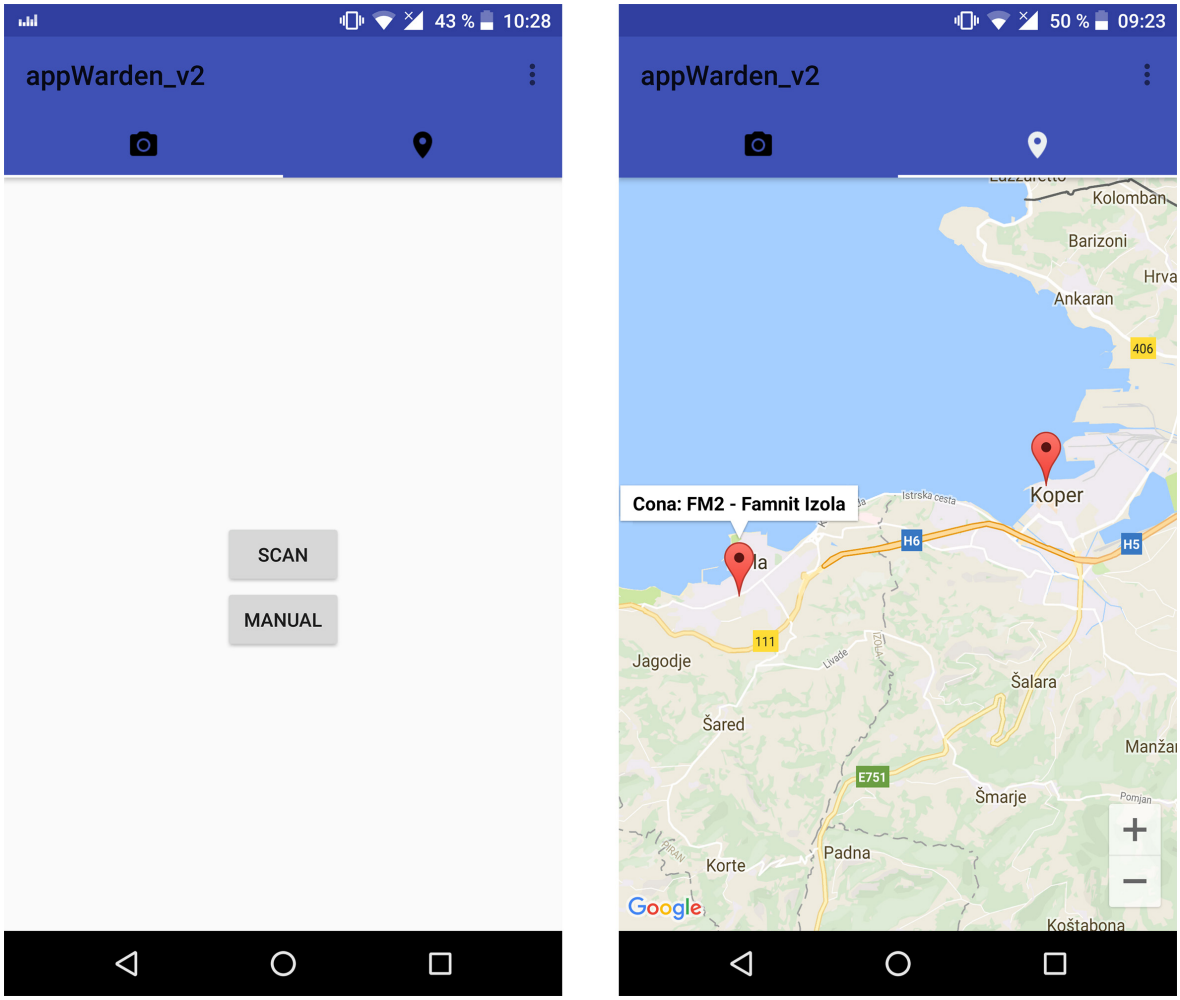
Uporabniški vmesnik je sestavljen iz devet zaslonov:

- **Zaslon za vpis:**
  Zaslon je namenjen vpisu v aplikacijo. Ima vnosno polje za uporabniško ime in za geslo. Na zaslonu bo tudi gumb s katerim potrdimo vnos.



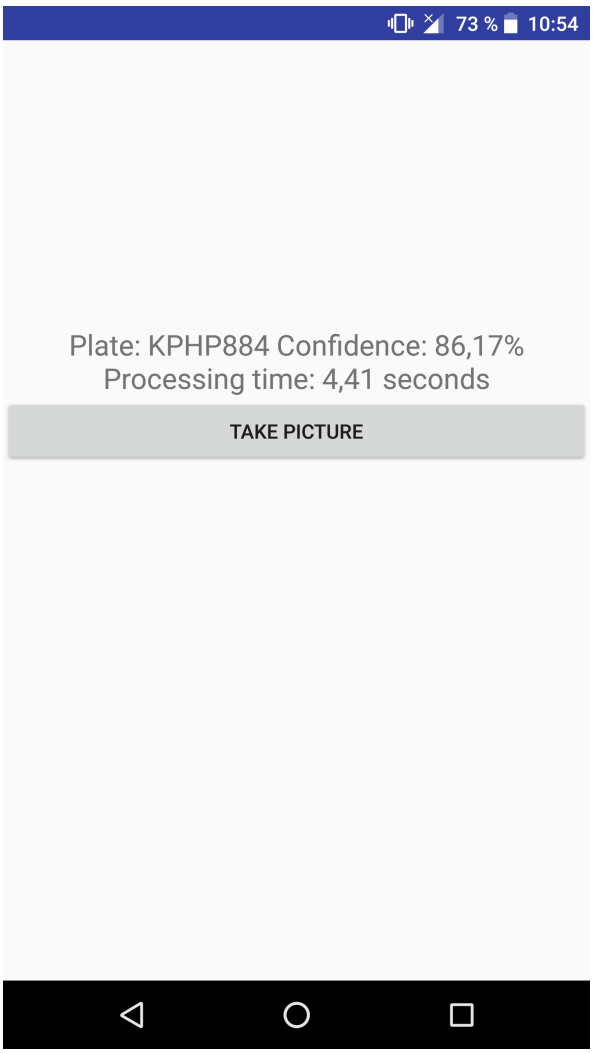
Slika 7: Zaslon za vpis.

- **Glavni zaslon:**
  Glavni zaslon je sestavljen iz dveh pod-zaslonov. Na prvem imamo gumbe za izbiro med iskanjem s kamero ali ročnim vnosom registrske tablice. Drugi pod-zaslon je v celoti namenjen prikazu zemljevida s parkirišči. Med pod-zasloni lahko prehajamo s potegom levo ali desno, ter s pritiskom na enega od znakov zaslona, ki se obarvajo belo v primeru, da je pod-zaslon izbran



Slika 8: Podzasloni glavnega zaslona.

- **Zaslon za izpis rezultatov in prižig kamere:**
  Ta zaslon je v celoti namenjen testiranju in je v kasnejši različici lahko preskočen oz. odstranjen. Na tem zaslonu je gumb s katerim prižgemo kamero ter tekstovno polje, kjer se izpišejo rezultati namenjeni izključno testiranju.

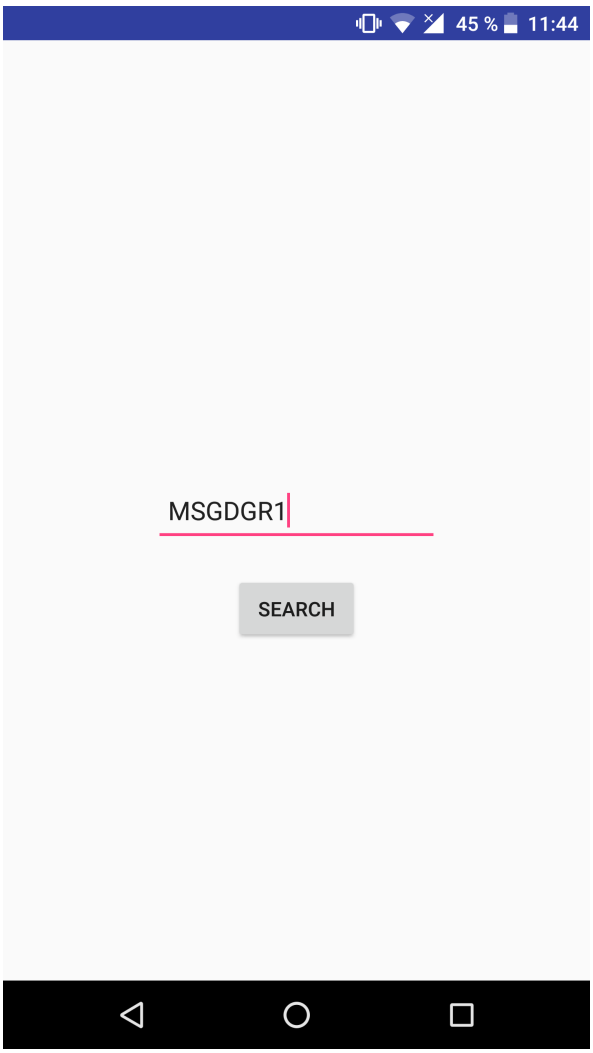


Slika 9: Zaslon za izpis rezultatov in prižig kamere.

- **Kamera:**
  Preprost uporabniški vmesnik, za Googlovo kamero, ki je vsebovan že v Androidu

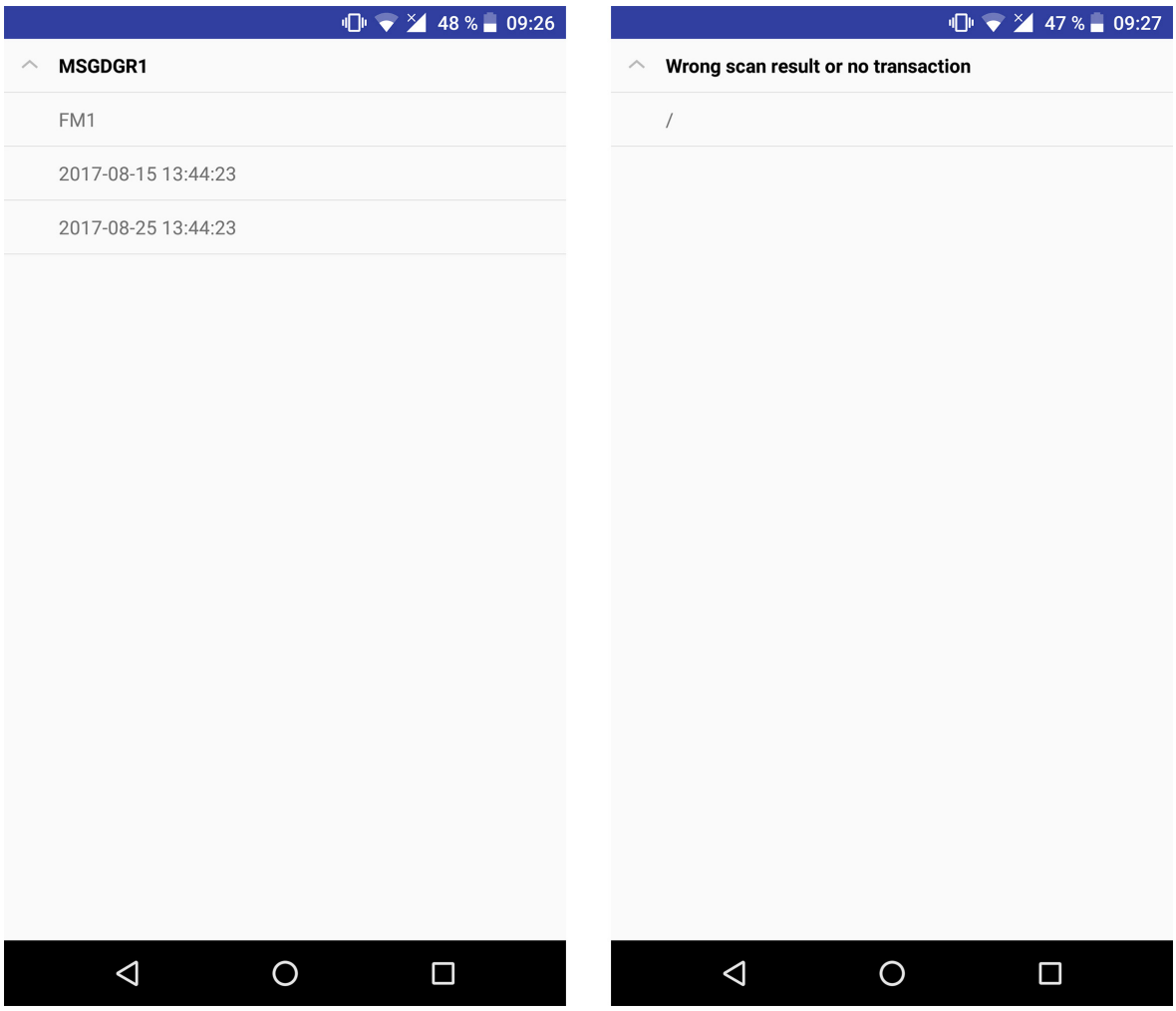
Slika 10: Uporabniški vmesnik Googlove kamerei.

- **Zaslon za ročni vnos:**
  Na zaslonu so le vnosno polje za registrsko tablico in gumb s katerim potrdimo vnos.



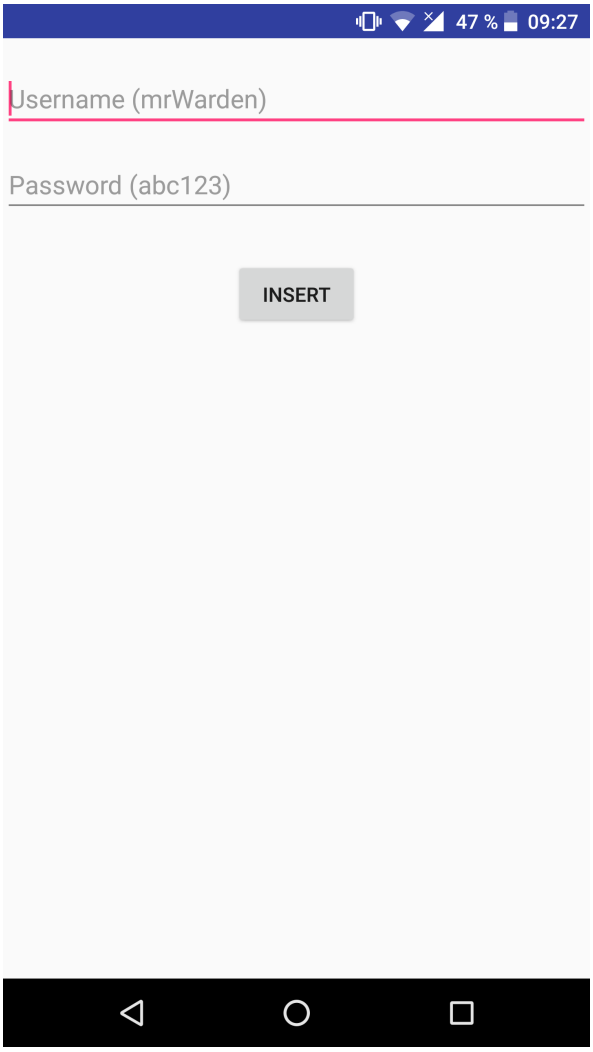
Slika 11: Zaslon za ročni vnos.

- **Zaslon za prikaz informacij o tablici:**
  Na zaslonu bo prikazano toliko padajočih seznamov, kolikor bo zadetkov za slikano registrsko tablico. V padajočem seznamu bodo zapisani vsi ključni podatki transakcije. V primeru, da ni zadetka bo seznam prazen.



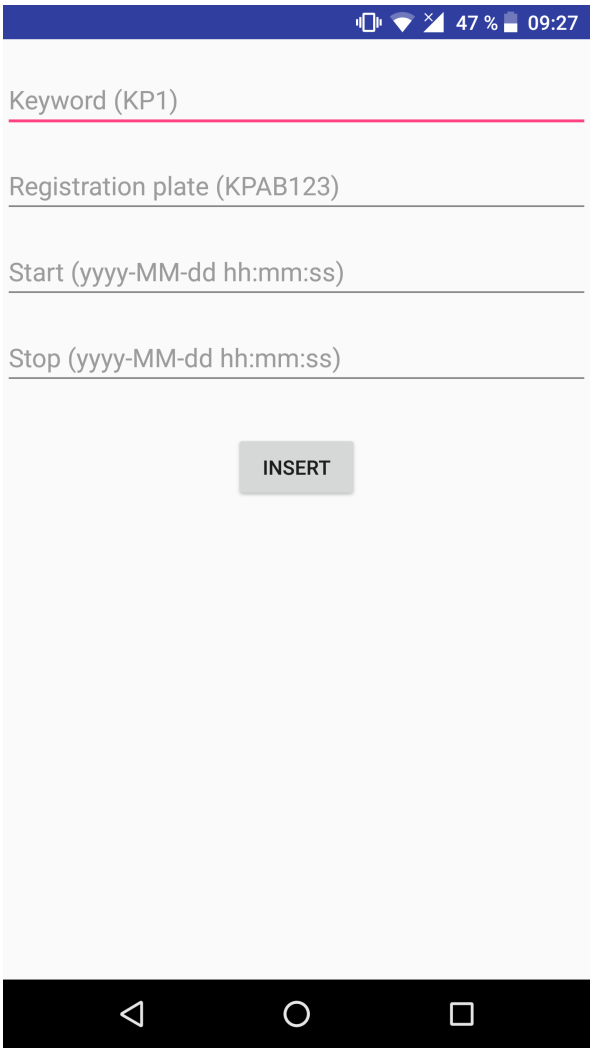
Slika 12: Zaslon za prikaz informacij o tablici.

- **Zaslon za vnos uporabnikov:**
  Zaslon ima vnosni polji za uporabniško ime in geslo, ter gumb za shranjevanje.



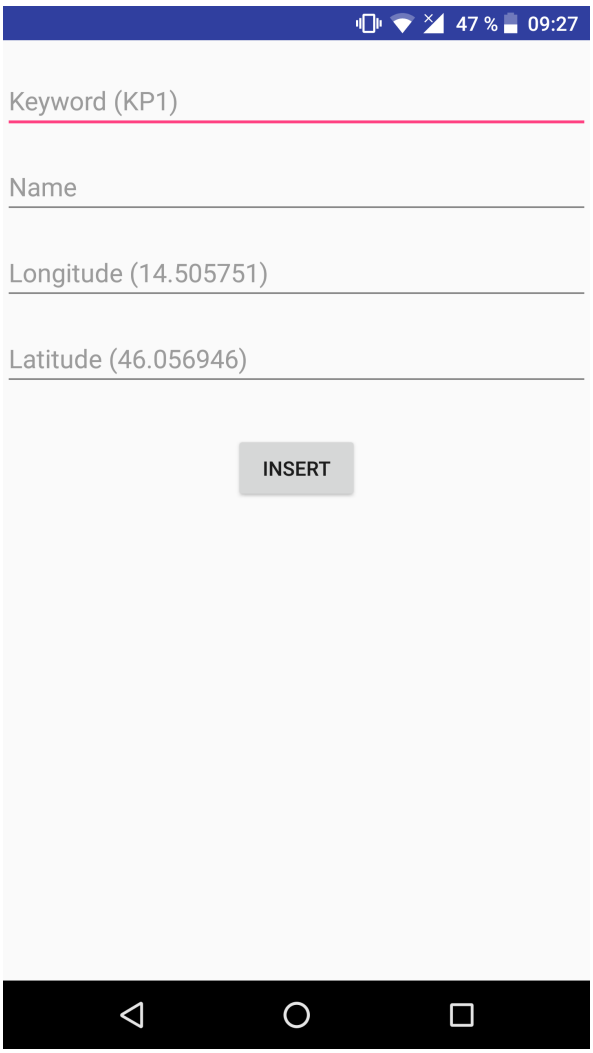
Slika 13: Zaslon za vnos uporabnikov.

- **Zaslon za vnos transakcij:**
  Na zaslonu je gumb za shranit in štiri vnosna polja za ključno besedo parkirišča, registrsko tablic, čas pričetka, čas zaključka parkiranja.



Slika 14: Zaslon za vnos transakcij.

- **Zaslon za vnos parkirišč:**
  Tudi ta zaslon ima gumb za shranit in štiri vnosna polja. Tako kot v prejšnjem ima tudi ta polje za ključno besedo parkirišča, poleg tega pa še polje za ime parkirišča in za geografsko širino in geografsko dolžino parkirišča.



Slika 15: Zaslon za vnos parkirišč.

5 Testiranje

Testiranje je bilo opravljeno z napravo Google Nexus 6P, na kateri je različica operacijskega sistema Android 8.0.0-beta, procesorsko moč mu zagotavlja Qualcomm Snapdragon 810, ter ima kamero z 12MP senzorjem. Ti podatki so pomembni, saj gre za napravo iz višjega cenovnega ranga, pri uporabnikih pa se pričakuje cenejše telefone z manj procesorske moči in slabšimi kamerami. Vse to pomeni, da bodo časi procesiranja slike na testni napravi hitrejši in rezultati natančnejši.

Ker gre pri registrskih tablicah za občutljive podatke, ni bilo mogoče slikati le-teh brez dovoljenja lastnikov, zato sem registrske tablice, za katere sem dobil dovoljenje, slikal večkrat, da sem dosegel mejo 100 slik.

5.1 Rezultati

Testiral sem na treh različnih oddaljenostih od registrske tablice in sicer na razdalji, ko je kamera ujela le tablico, nato tako, da je bilo vidno še malo okolice in nazadnje še tako, da je bil v kadru celoten zadnji del avta. Telefon sem poskušal držati pod pravim kotom glede na tablico in vse slike so bile narejene v optimalnih svetlobnih pogojih.

Rezultati so bili naslednji:

Tabela 1: Rezultati testiranja

	Povprečen čas	Uspešnost
Test - samo tablica	7s	85%
Test - tablica + okolica	12s	80%
Test - celoten zadnji del avta	29s	65%

Iz zgornje tabele lahko razberemo, da najbolj optimalno zajemanje slik je takrat, ko je v kadru samo tablica. Tak rezultat lahko pripišemo temu, da na sliki ni ostalih motečih dejavnikov in je samo iskanje in prepoznavanje registrske tablice hitreje in natančnejše.

5.2 Pomanjklivosti in možne izboljšave

Med samim testiranjem so prišle do izraza tudi določene pomanjklivosti prototipa oz. procesiranja slik. Napaki, ki sta se najpogosteje pojavljali sta bili, zamenjava črke O z številko 0 in obratno, ter zaznava roba tablice kot 1 ali I. Rešitev, s katerim bi se ti napaki odpravilo oz. vsaj izboljšalo, je učenje OCR algoritma, ki pa zaradi časovne omejenosti pri projektu, to ni bilo mogoče.

Nadaljnje izboljšave bi se lahko naredilo tudi na področju časovne zahtevnosti procesiranja slike. Možna izboljšava za procesorski čas, bi bil prenos procesiranja slik na strežnik, saj je procesorska moč neprimerljivo boljša. To bi bilo izvedljivo v primeru, da bi se kakšen od ponudnikov elektronskega parkiranja odločil za uporabo tega prototipa.

6 Zaključek

V zaključni nalogi je bil predstavljen postopek razvoja prototipne rešitve za prepoznavanje registrskih tablic s pomočjo pametne naprave Android. Iz pregleda obstoječih rešitev, smo ugotovili njihove prednosti in slabosti. Poleg tega smo ugotovili tudi, da so te zelo drage in ekonomsko neupravičene, saj je z idejo prototipa dosežemo iste rezultate.

Opisana je celotna izdelava prototipa, tako zasnova kot implementacija prototipne aplikacije. Aplikacija za prepoznavanje registrskih tablic uporablja OpenALPR knjižnico, ki je prav tako predstavljena v zaključni nalogi. Ker gre za prototipno aplikacijo in ni integrirana z že obstoječo bazo ponudnikov elektronskega plačevanja parkirnine, jo je bilo potrebno prilagoditi, zato ima aplikacija nekaj posebnosti.

Opravljeno je bilo tudi testiranje aplikacije. Iz vzorca 100 slik so povzete ugotovitve in predstavljene so pomanjklivosti in možne izboljšave.

V prihodnosti vidim potencial združitve prototipa z že obstoječim sistemom podjetja, ki ponuja rešitve elektronskega parkiranja. To bi omogočilo razvoj vseh potencialov take aplikacije, saj bi projekt imel večji ekonomski vložek, posledično boljše pogoje za razvoj.

7 Literatura

[1] *ACEA – Passenger car registrations: +5.3% five months into 2017; +7.6% in May*, http://www.acea.be/press-releases/article/passenger-car-registrations-5.3-five-months-into-2017-7.6-in-may. (Datum ogleda: 11. 7. 2017.) *(Citirano na strani 1.)*

[2] *What is Java technology and why do I need it?*, https://www.java.com/en/download/faq/whatis_ java.xml. (Datum ogleda: 11. 7. 2017.) *(Citirano na strani 9.)*

[3] Sam Deering, *Why use JSON over XML?*, https://www.sitepoint.com/json-vs-xml/. (Datum ogleda: 1. 8. 2014.) *(Citirano na strani 5.)*

[4] *Why is Everyone Choosing JSON Over XML for jQuery?*, https://stackoverflow.com/questions/1743532/why-is-everyone-choosing-json-over-xml-for-jquery. (Datum ogleda: 1. 8. 2017.) *(Citirano na strani 5.)*

[5] *Everything you need to build on Android*, https://developer.android.com/studio/features.html. (Datum ogleda: 1. 8. 2017.) *(Citirano na strani 9.)*

[6] Lars Vogel, *Getting started with Android development - Tutorial*, http://www.vogella.com/tutorials/Android/article.html. (Datum ogleda: 3. 8. 2014.) *(Citirano na strani 9.)*

[7] *OpenALPR Design*, http://doc.openalpr.com/opensource.html# openalpr-design. (Datum ogleda: 9. 8. 2017.) *(Citirano na strani 10.)*

[8] Abdallah A. Mohamed in Roman V. Yampolskiy, An Improved LBP Algorithm for Avatar Face Recognition . 2011, poslano v objavo. *(Citirano na strani 11.)*

[9] *Binarization*,
http://doc.openalpr.com/opensource.html# binarization. (Datum ogleda: 9. 8. 2017.) *(Citirano na strani 11.)*

[10] *Thresholding (image processing)*,
https://en.wikipedia.org/wiki/Thresholding_ (image_ processing). (Datum ogleda: 9. 8. 2017.) *(Citirano na strani 11.)*

[11] *Character-Analysis*,
http://doc.openalpr.com/opensource.html# character-analysis. (Datum ogleda: 9. 8. 2017.) *(Citirano na strani 12.)*

[12] *Plate-Edges*,
http://doc.openalpr.com/opensource.html# plate-edges. (Datum ogleda: 9. 8. 2017.) *(Citirano na strani 12.)*

[13] *Deskew*,
http://doc.openalpr.com/opensource.html# deskew. (Datum ogleda: 9. 8. 2017.) *(Citirano na strani 12.)*

[14] *Character-Segmentation*,
http://doc.openalpr.com/opensource.html# character-segmentation. (Datum ogleda: 9. 8. 2017.) *(Citirano na strani 12.)*

[15] *OCR*,
http://doc.openalpr.com/opensource.html# ocr. (Datum ogleda: 10. 8. 2017.) *(Citirano na strani 13.)*

[16] *Optical character recognition*,
https://en.wikipedia.org/wiki/Optical_ character_ recognition. (Datum ogleda: 10. 8. 2017.) *(Citirano na strani 13.)*

[17] *Post-Processing*,
http://doc.openalpr.com/opensource.html# post-processing. (Datum ogleda: 10. 8. 2017.) *(Citirano na strani 13.)*

[18] *Quick guide to getting a key*,
https://developers.google.com/maps/documentation/android-api/signup. (Datum ogleda: 10. 8. 2017.) *(Citirano na strani 13.)*

[19] *Building for Scale: Updates to Google Maps APIs Standard Plan*,
https://maps-apis.googleblog.com/2016/06/building-for-scale-updates-to-google.h (Datum ogleda: 10. 8. 2017.) *(Citirano na strani 13.)*

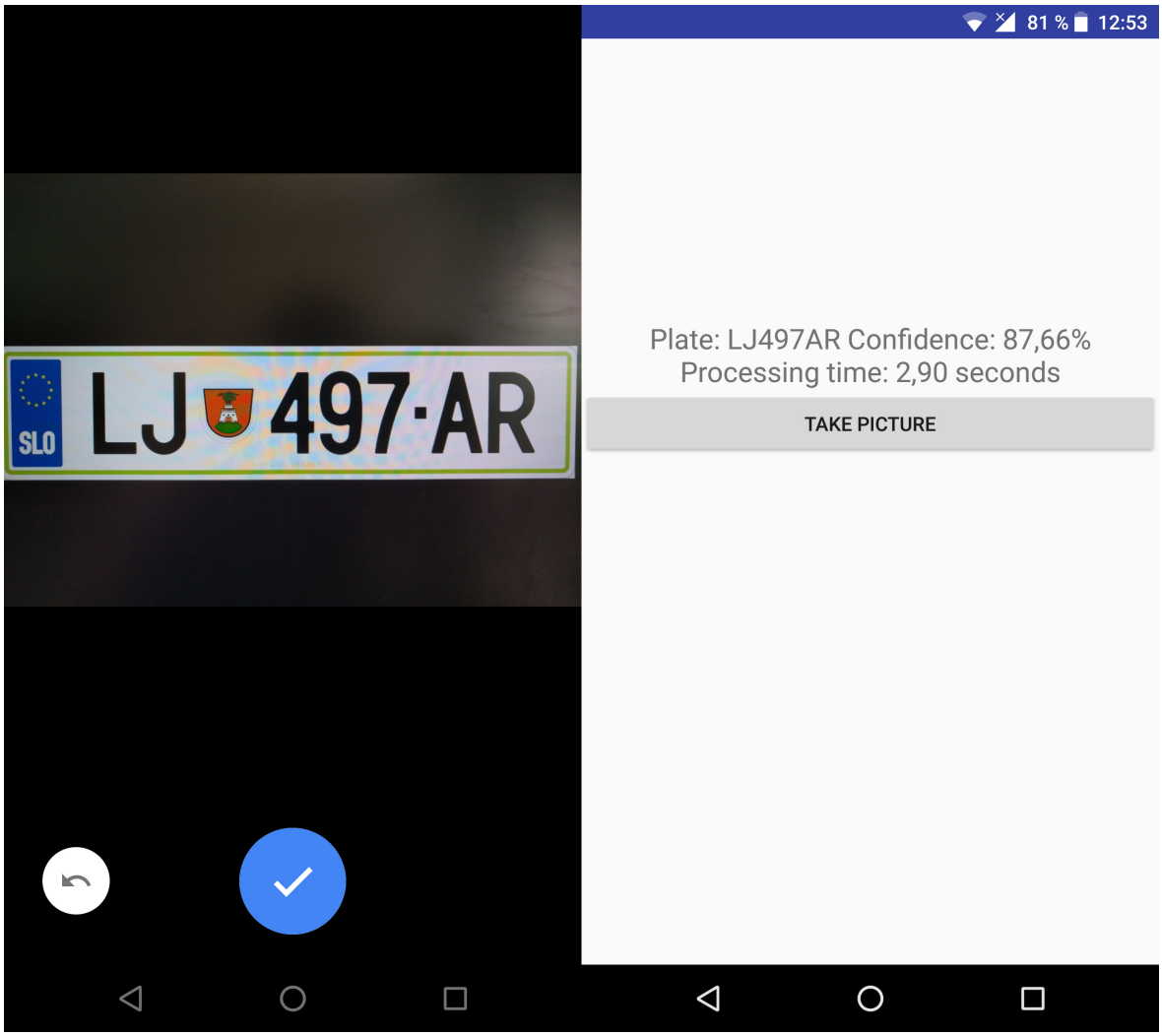
[20] *The Realm Mobile Platform*,
`https://realm.io/docs/get-started/overview/`. (Datum ogleda: 12. 8. 2017.)
(Citirano na strani 15.)

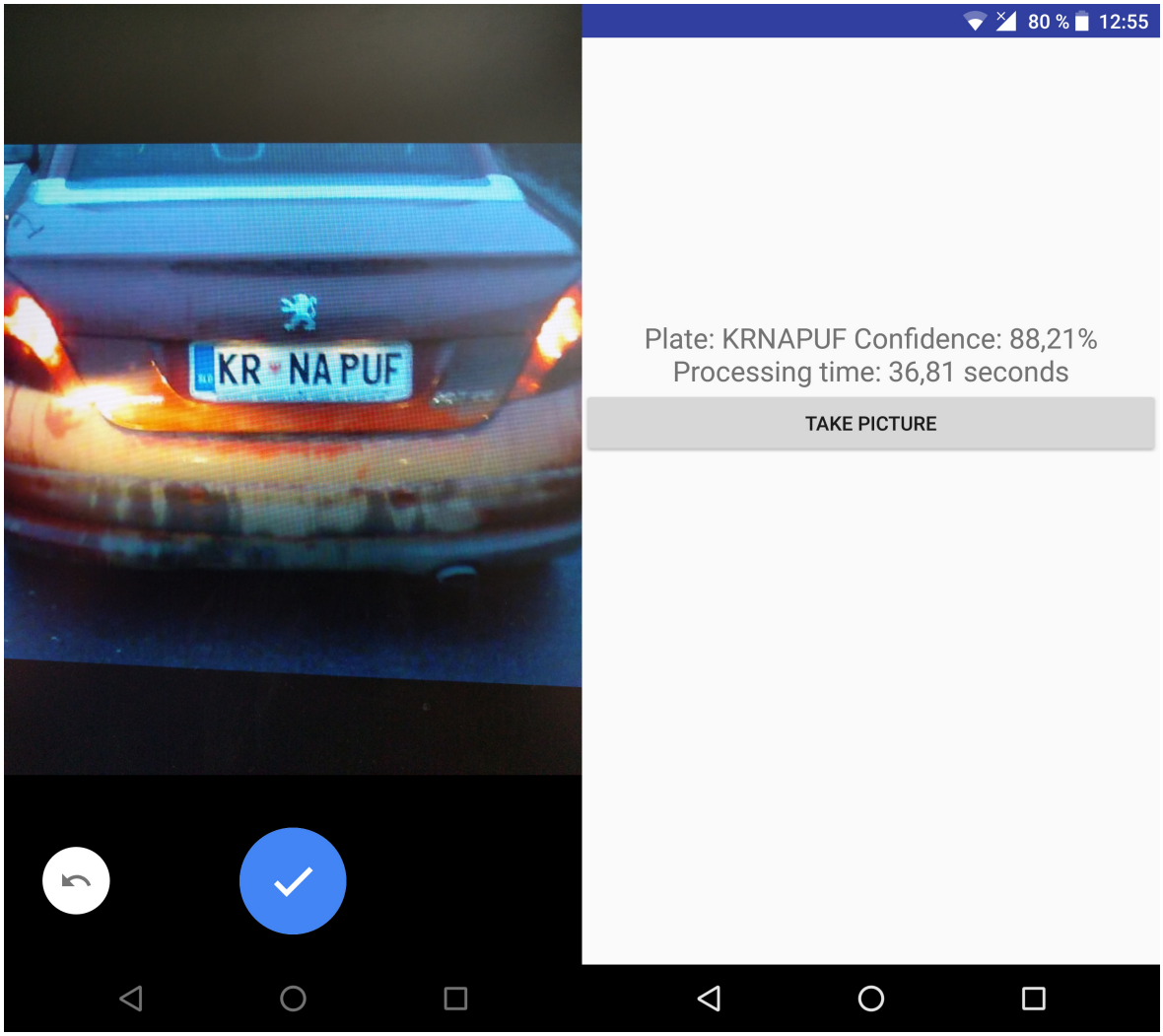
Priloge

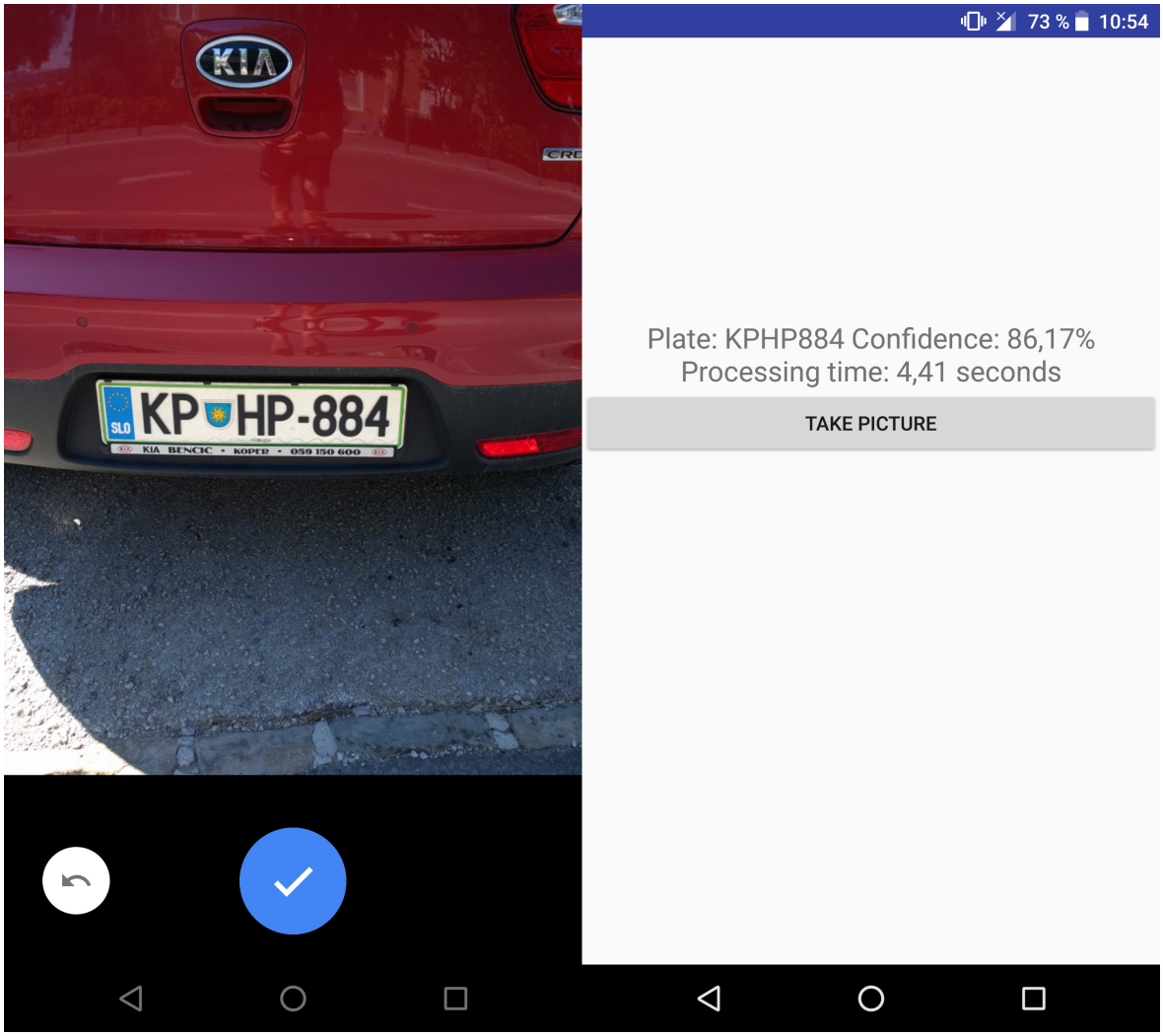
A Izvorna koda

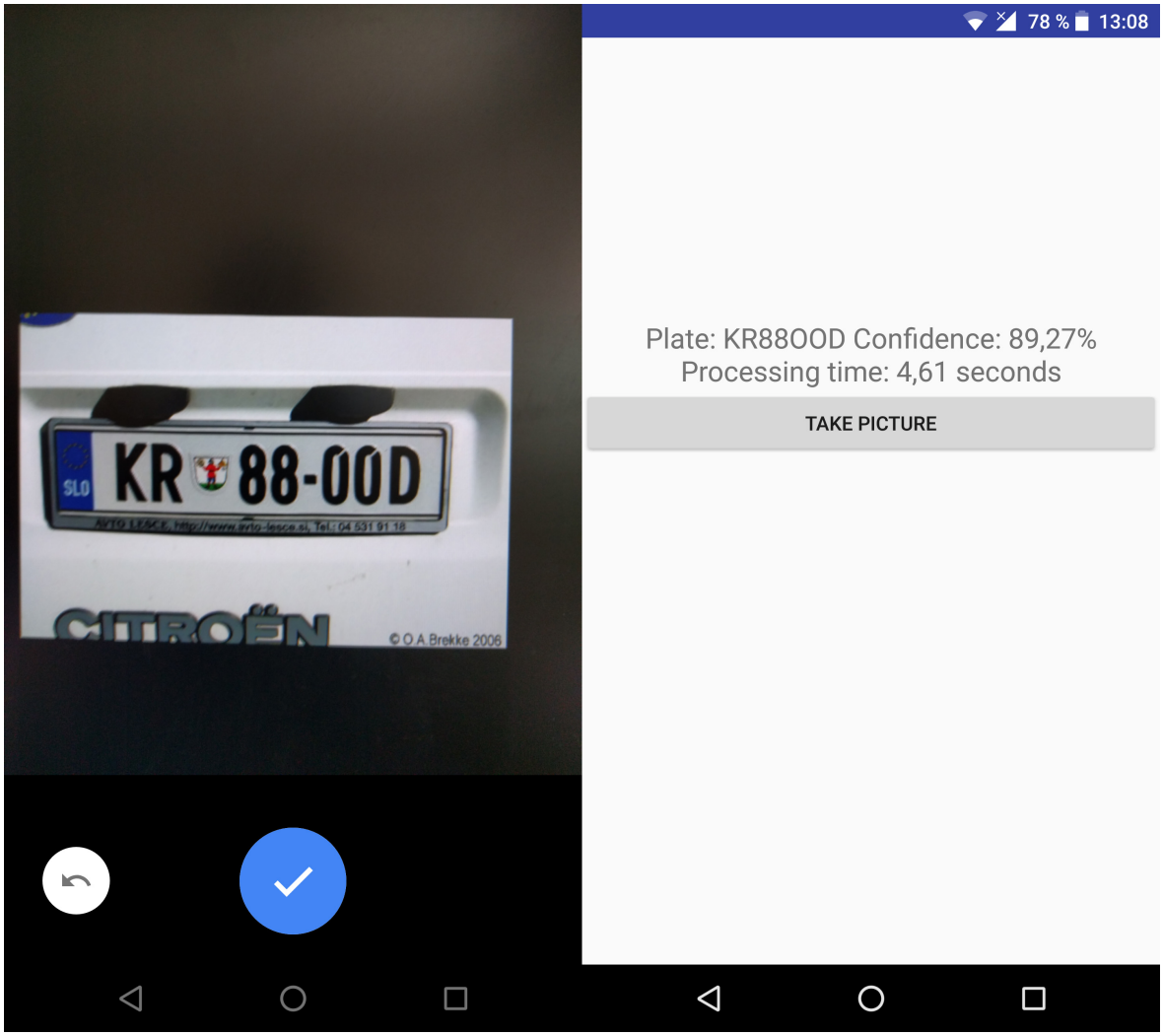
Priložena je zgoščenka z vso izvorno kodo aplikacije.

B Primeri testiranja




80 % 12:55
KR NA PUF
Plate: KRNAPUF Confidence: 88,21%
Processing time: 36,81 seconds
TAKE PICTURE


KIA
KP HP-884
KIA BENČIČ • KOPER • 059 150 600
73 % 10:54
Plate: KPHP884 Confidence: 86,17%
Processing time: 4,41 seconds
TAKE PICTURE


78 % 13:08
KR 88-00D
SLO
CITROËN
© O.A Brekke 2006
Plate: KR88OOD Confidence: 89,27%
Processing time: 4,61 seconds
TAKE PICTURE