SVEUČILIŠTE U SPLITU

**PRIRODOSLOVNO MATEMATIČKI FAKULTET**

**Expense tracker**

**Dokumentacija**

|  |  |
| --- | --- |
| **Studij:** | Informatika |
| **Predmet:** | Upravljanje znanjem u razvoju programske podrške, Programiranje mobilnih aplikacija |
| **Ak.god.:** | 2018/2019 |
| **Profesor:** | Saša Mladenović |
| **Studentica:** | Christina Paula Glušević |

Split, veljača 2019.

Sadržaj

[Uvod 1](#_Toc1147499)

[1. Opis sustava 2](#_Toc1147500)

[1.1. Model sustava 3](#_Toc1147501)

[2. Razvoj sustava 4](#_Toc1147502)

[2.1. Registracija i login 4](#_Toc1147503)

[2.2. Unos novog troška 5](#_Toc1147504)

[2.3. Prikaz i brisanje troškova 7](#_Toc1147505)

[3. OCR 8](#_Toc1147506)

[3.1. Tesseact.js 8](#_Toc1147507)

[3.1.1. Metoda crne kutije 9](#_Toc1147511)

[3.2. Rezultati 10](#_Toc1147512)

[3.3. Dohvaćanje podataka iz rezultata 13](#_Toc1147513)

[4. Zaključak 14](#_Toc1147514)

[Bibliografija 15](#_Toc1147515)

# Uvod

Za projekt iz predmeta Programiranje mobilnih aplikacija i Upravljanje znanjem u razvoju programske podrške izrađena je sustav koji korisnicima omogućava praćenje vlastitih troškova. Prije samog korištenja aplikacije korisnici se moraju registrirati ili logirati, a zatim mogu unositi svoje troškove, pregledati ih ili izbrisati. Unošenje troškova moguće je na dva načina, korisnik može samostalno unijeti podatke ili može slikati račun. Kada se koristi druga opcija, slika se šalje na obradu koja uključuje optičko prepoznavanje znakova (engl. *Optical Character Recognition*) i izvlačenje potrebnih podataka iz računa.

Projekt se sastoji od dva dijela, mobilne aplikacije izrađene koristeći html, JavaScript i CSS u okviru za razvoj mobilnih aplikacija PhoneGap te servera napravljenog u Node.js-u, serverskoj Javascript platformi.

U ovoj dokumentaciji prikazat će se kako aplikacija radi, koji su problemi i moguće preinake za poboljšanje učinkovitosti OCR-a.

# Opis sustava

Svakodnevno trošimo novac na različite načine, a tehnologija nam omogućava da lakše pratimo vlastite financije. Iz tog razloga odabrala sam napraviti aplikaciju kojom jednostavno možemo voditi dnevnik troškova.

Prije same izrade sustava potrebno je razraditi model te definirati kako bi sustav trebao raditi. Sama aplikacija korisniku treba omogućiti:

* Registraciju
* Login
* Odjavu
* Ručni unos novog troška
* Unos novog troška putem slike
* Pregled troškova
* Brisanje troškova.

Koristeći PhoneGap okvir jednostavno je realizirati aplikaciju kao single page aplikaciju koja zadovoljava gore navedene zahtjeve.

Server prima zahtjeve od klijenta, odnosno aplikacije i po primljenom zahtjevu server treba:

* Pohraniti novog korisnika
* Vratiti odgovor pri loginu korisnika
* Pohraniti unesene podatke
* Obraditi primljenu sliku za dohvaćanje potrebnih podataka
* Poslati popis troškova korisnika.

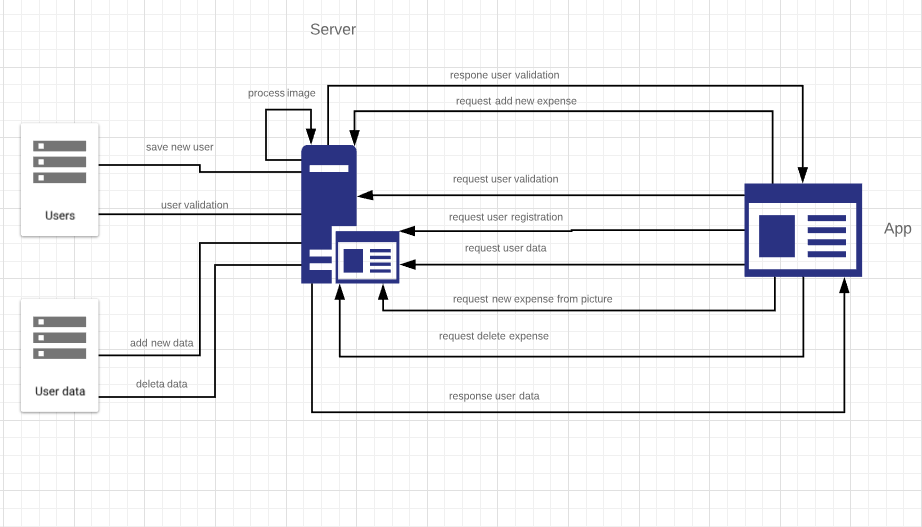
Entiteti su sve što se može jednoznačno odrediti, identificirati i razlikovati te on može biti realan ili apstraktan objekt. Najvažniji entiteti u ovom projektu su korisnik i trošak.

**Korisnik** – svaki korisnik ima svoj jedinstven id, ime, e-mail adresu i lozinku. Pri registraciji korisnik unosi svoje ime, e-mail i lozinku, a id se automatski dodjeljuje.

**Trošak –** trošak se definirao kao objekt koji se sastoji od datuma, naziva i vrijednosti. Prilikom unošenja novog troška datum i naziv su izborne vrijednosti, ukoliko ih korisnik ne unese dodjeljuje se današnji datum za datum i „*Undefined“* za naziv.

# Model sustava

Model sustava je pojednostavljeni prikaz dijela svijeta koji se koristi za komunikaciju, dokumentaciju, analizu, simulaciju, donošenje odluka i provjeru. Slika 1. prikazuje apstraktni model projekta.



Slika 1Model sustava

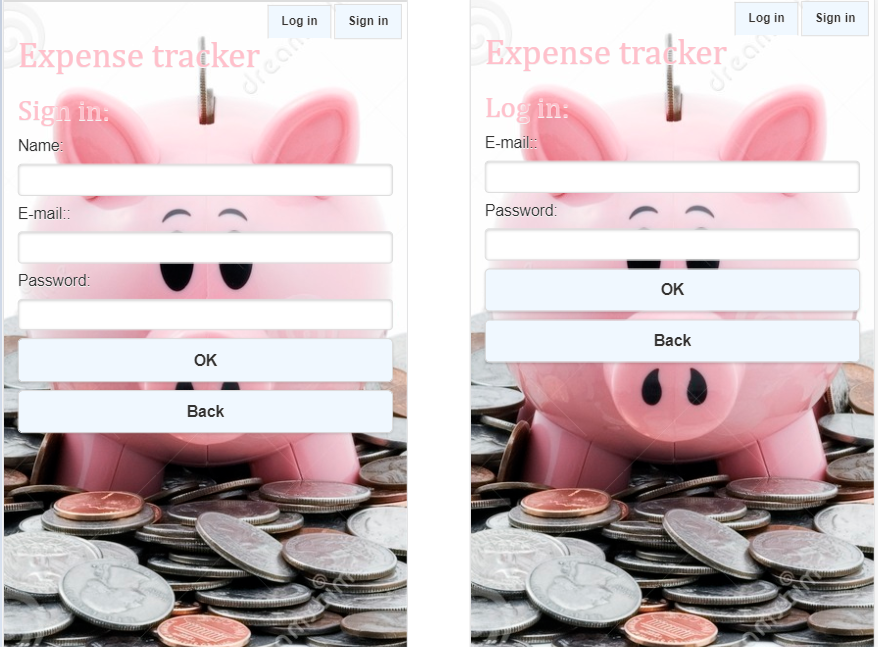
Kao što je vidljivo iz slike, korisnik kroz aplikaciju šalje zahtjeve serveru, zahtjevi koje aplikacija može slati odgovaraju radnjama koje korisnik može vršiti, registracija, login, unos novog troška, prikaz troškova i brisanje troškova. Server aplikaciji šalje odgovore, odnosno on potvrđuje registraciju ili login te šalje pohranjene korisničke podatke. Server je također zadužen za obradu slika.

# Razvoj sustava

Redoslijed izrade projekta odgovara načinu korištenja aplikacije. Implementirala sam redom one dijelove sustava koji odgovaraju redoslijedu koje korisnik mora slijediti da bi mogao koristiti sustav.

# Registracija i login

Prvi korak s kojim se korisnik susreće pri korištenju aplikacije je registracija. Jednostavno sučelje od korisnika zahtjeva unos imena, e-mail-a i lozinke. Slično je i za login, ali pri loginu dovoljno je unijeti ispravan e-mail i lozinku.



Slika 2. Registracija i login

Nakon što korisnik unese svoje podatke šalje se zahtjev serveru koji zatim provjerava dobivene podatke. Pri registraciji provjerava se je li se uneseni mail koristi i u tom slučaju obavještava se korisnik. Kod logina server provjerava e-mail adresu te podudaranje lozinke.

# Unos novog troška

Kao što je gore navedeno, trošak se sastoji od datuma, naziva i vrijednosti. Korisniku se nudi mogućnost ručnog unosa tih vrijednosti ili slikanja računa pri čemu se nad njim vrši obrada da bi se dohvatile te vrijednosti, koje korisnik zatim samo potvrđuje.



Slika 3.Slanje zahtjeva

Aplikacija sa serverom komunicira slanjem XMLHttpRequest-a. XMLHttpRequest je objekt koji se koristi za razmjenu podataka. U ovom slučaju šaljemo podatke preko „*query-a“*. Na slici 3. vidimo dohvaćanje unesenih podatak i slanje zahtjeva.

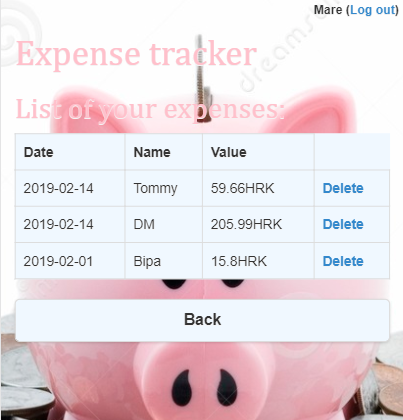


Slika 4. Spremanje podataka

Slika 4. prikazuje pohranu podataka po primljenom zahtjevu. Podaci se spremaju u CSV datoteku koju ima svaki registrirani korisnik. Slično se događa i kada korisnik želi unijeti podatke preko slike, o tome će pisati detaljnije u idućem poglavlju.

# Prikaz i brisanje troškova

Da bi korisnik mogao upravljati svojim troškovima potrebno je imati uvid u njih. Slično kao i što se podaci pohranjuju, mogu se i dobavit slanjem XMLHttpRequest-a, ali ovoga puta šaljemo samo id korisnika da bi se mogli dohvatiti ispravni podaci. Podaci se prikazuju u tablici te je uz svaki trošak dodana opcija brisanja.



Slika 5. Lista troškova

Klikom na *„Delete“* dugme šalje se zahtjev na server koji briše odabrani trošak, a zatim vraća osvježenu listu troškova.

# OCR

Jedna od mogućnosti u izrađenom sustavu je unošenje troškova preko slike. Da bi to bilo moguće potrebno je izvući potrebne podatke iz slike, a za to se koristilo optičko prepoznavanje znakova.

Optičko prepoznavanje znakova je mehanička ili elektronička pretvorba tiskanog ili rukopisnog teksta u strojno kodiran tekst koja omogućava digitalizaciju različitih dokumenata. OCR možemo promatrati kao strojnu replikaciju ljudskih funkcija koja se u posljednjih pet desetljeća razvila do jedne od najuspješnijih tehnologija u području prepoznavanja uzoraka i umjetne inteligencije. U posljednjih par godina OCR tehnologije značajno su napredovale (prema nekim rezultatima do 99% točnosti), ali još uvijek nije jednako ljudskom čitanju. Problem mogu stvarati različiti faktori poput buke (engl. *noise*) u slici ili dokumentu, „slomljeni“ ili „razmazani“ znakovi i slično.

Postupak dobivanja teksta iz slike ili tiskanog dokumenta odvija se u više koraka koji uključuju:

* Binarizaciju – pretvaranje slike u crno bijelu sliku radi lakšeg prepoznavanja
* Analiza dokumenta ili slike – prepoznavanje zona interesa, odnosno označavanje dijelova koji sadrže tekst
* Segmentacija linija – podjela dokumenta prema linijama teksta
* Segmentacija riječi ili znakova – odnosi se na izoliranje pojedinih riječi ili znakova u liniji
* Prepoznavanje znakova – prepoznavanje karakteristika svakog znaka i dodjeljivanje simbola

# Tesseact.js



U ovom projektu koristio se Tesseract.js, JavaScript biblioteka koja omogućava korištenje Tesseract mehanizma za prevođenje više od 60 jezika. Tesseract.js uključuje automatsku detekciju orijentacije teksta, čitanje paragrafa, riječi i znakova te se može koristiti u pregledniku ili na serveru koristeći Node.js. U ovom projektu Tesseract se koristio na strani servera. Postavljanje je jednostavno, sve što treba napraviti je preko npm-a instalirati Tesseract i uključiti ga u projekt pomoću

var Tesseract = require('tesseract.js') ,

i Tesseract je spreman za rad. Za pokretanje obrade slike pozivamo metodu recognize kojoj prosljeđujemo lokaciju slike.

Tesseract.recognize(filename)

.progress(function (p) { console.log('progress', p) })

.catch(err => console.error(err))

.then(function (result) {

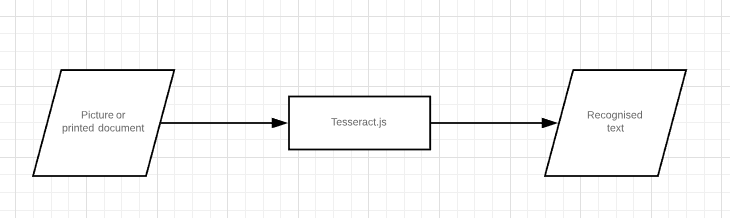
console.log(result.text)

process.exit(0)

})

# Metoda crne kutije

U znanosti, računalstvu i inženjerstvu metoda crne kutije može se definirati kao uređaj, sustav ili objekt koji promatramo u kontekstu ulaza i izlaza, bez ikakvog znanja o njegovim unutarnjim dijelovima. Na ovaj način je u ovom projektu korišten Tesseract.js.

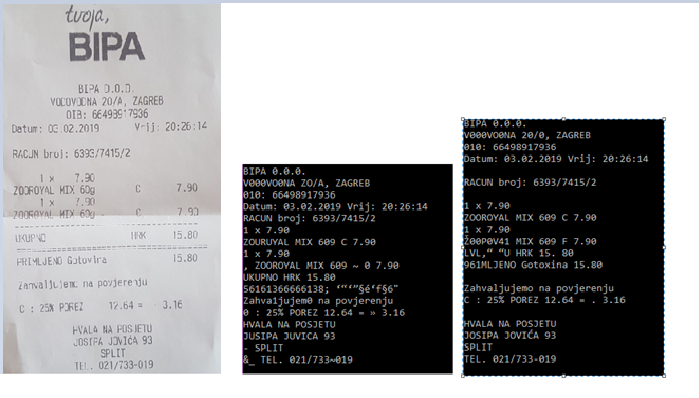


Slika 6. Tesseract.js

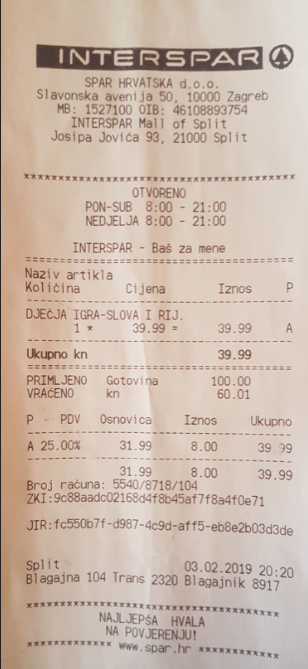
Potrebno je napomenuti da Tesseract.js koristi već uvježbanu mrežu. Podaci na kojima je mreža trenirana najčešće su različiti paragrafi teksta, a u ovom projektu koristili su se računi koji su drugačije strukturirani pa rezultati nisu uvijek najbolji.

# Rezultati

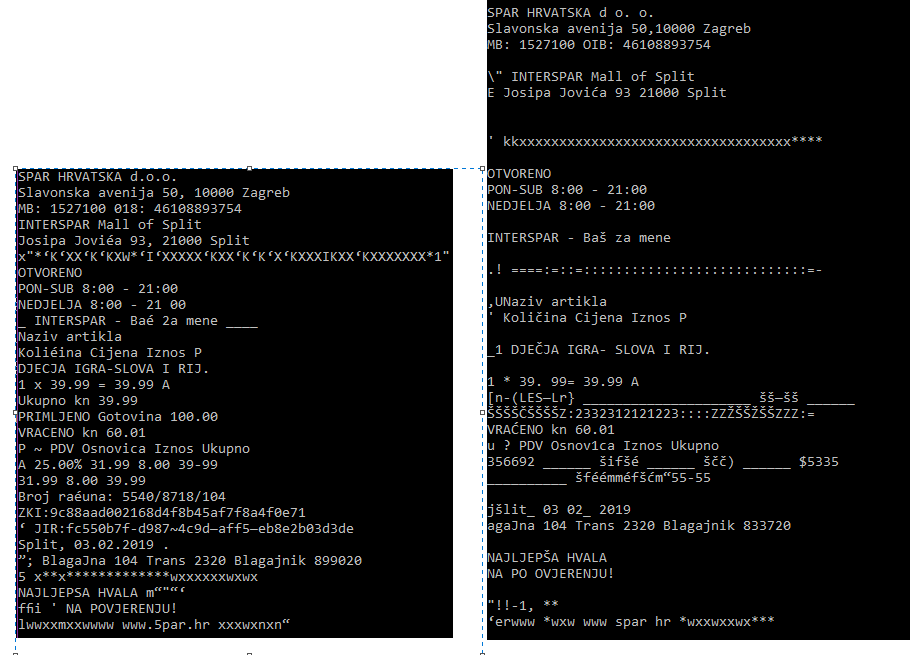
Kako bi se pronašlo najbolje rješenje isprobala sam tesseract.js i node-tesseract biblioteke te ih testirala na različitim računima. Rezultati nisu uvijek najbolji, ponekad nisu ni zadovoljavajući. Kriterij odabira završne verzije je bio uspješnost u prepoznavanju datuma, naziva dućana ili tvrtke koja je izdala račun i ukupne cijene. Iako postoji još npm biblioteka za OCR one se plaćaju ili nude ograničeni broj obrada pa nisu korištene.



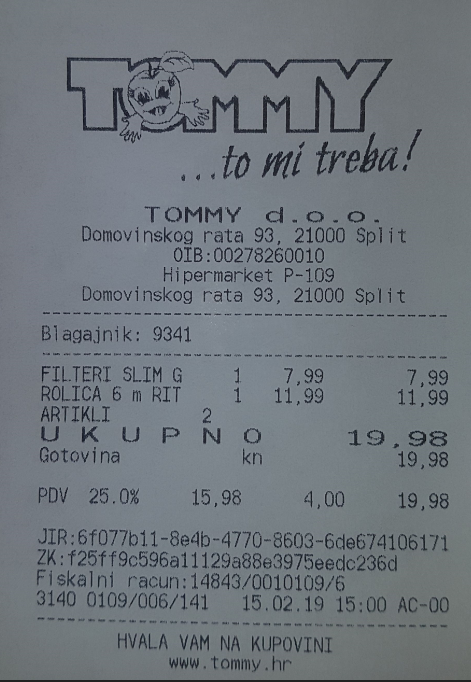
Slika 7. Originalan račun / tesseract.js / node-tesseract



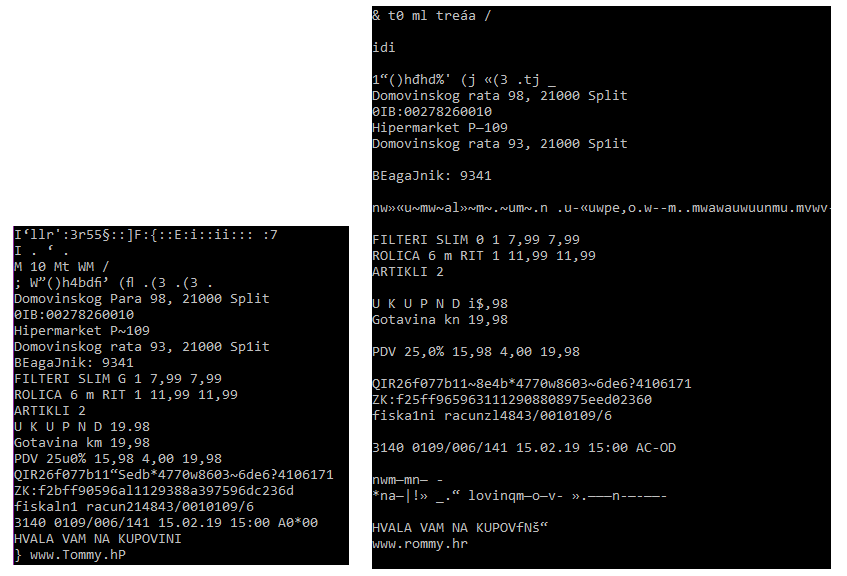
Slika 8.Original račun



Slika 9. tesseract.js / node-tesseract



Slika 10. Originalan račun



Slika 11.tesseract.js / node-tesseract

Ako promotrimo slike, na prva dva računa prepoznate su sve potrebne stavke (datum, naziv i vrijednost), a na posljednjem računu nije prepoznat naziv „Tommy“ i node-tesseract nije dobro prepoznao ukupnu cijenu. Ipak, nešto bolje rezultate daje tesseract.js, pa sam se odlučila za njega. Problem predstavljaju različiti fontovi, buka slike i isprekidanost slova, zgusnutost znakova i slično. U slučaju da se ne uspiju prepoznati sve ili neke od stavki korisniku je otvorena mogućnost da se podaci ručno unesu prije pohrane podataka.

# Dohvaćanje podataka iz rezultata

Sada kada imamo rezultate OCR potrebno je izvući podatke koji nas zanimaju. Da bi to ostvarila, koristila sam regularne izraze. Regularni izrazi (regex) su nizovi znakova koji definiraju određeni izraz. Regularni izrazi koriste se najčešće za pretraživanje određenog podatka u nekom tekstu ili pri validaciji korisničkog unosa.

Korišteni su regularni izrazi za pronalaženje datuma i cijene.

Za datum tražimo brojeve koji su napisani u obliku dd.mm.gggg, koristeći slijedeći regularni izraz:

/([0-3]?\d(\s|\.|\s){1})([01]?\d(\s|\.|\s){1})([12]{1}\d{3}\.?)/g

Dakle, regularni izraz nam govori da prva znamenka može biti 0,1,2,3, nakon nje slijedi nula ili jedna brojčana znamenka, zatim slijedi „.“ ili razmak, pa se slično ponavlja i za mjesec i godinu.

Za dohvaćanje ukupne cijene koriste se dva načina. Promatrajući račune vidimo da se ukupan iznos računa uvijek nalazi u redu zajedno sa riječi „ukupno“, pa prvi način radi tako da pronađe redak s riječi ukupno, a zatim se iz tog retka pronalazi cijena. Koriste se slijedeći regularni izrazi:

*  za pronalaženje retka s riječi ukupno ili ukup
*  za pronalaženje cijene u retku.

Drugi način koji je korišten je dohvaćanje svih brojčanih vrijednosti koje odgovaraju obliku cijene (jedan ili više broj s lijeve strane, zatim točka ili zarez te dva brojčana znaka:

.

Kada se koristi drugi način dobivamo niz decimalnih brojeva iz kojeg za vrijednost uzimamo najveći.

Nakon što se dohvate svi potrebni podaci server ih šalje u odgovoru, a korisniku se u aplikaciji automatski ispune polja te mu ostaje potvrditi unos prije spremanja.

# Zaključak

Sustav bi bilo dobro nadograditi na način da se korisniku omogući unos i pregled detalja svakog troška. Koristeći ručni unos korisniku bi moglo biti naporno unositi sve detalje, a prilikom automatskog unosa dolazimo do problema s OCR-om. Iako tesseract.js daje dobre rezultate, ponekad su nezadovoljavajući pa je upitno kako bi sustav funkcionirao dodavanjem detalja. Još jedan problem pri korištenju tesseract.js-a je vrijeme izvršavanja. Naime, nakon što korisnik slika račun potrebno je pričekati neko vrijeme da bi server vratio rezultate. Također, tesseract.js koristi već utreniranu mrežu što u ovom slučaju nije poželjno. Bilo bi dobro kada bi se mrežu moglo trenirati na različitim računima, time bi dobivali bolje rezultate.

# Bibliografija

|  |  |
| --- | --- |
| [1] | S. Mladenović i K. Klarin, Upravljanje znanjem u razvoju programske podrške, 2014/2015. |
| [2] | S. G. Dafe i S. S. Chavhan, Optical Character Recognition Using Image Processing, 2018. |
| [3] | Automating Receipt Processing, 2017. |