UNIVERZA NA PRIMORSKEM
FAKULTETA ZA MATEMATIKO, NARAVOSLOVJE IN
INFORMACIJSKE TEHNOLOGIJE

Zaključna naloga

Mobilna aplikacija za odkrivanje škodljivih aditivov v živilskih izdelkih

(Mobile application for detecting harmful food additives in food products)

Ime in priimek: Matej Brlec
Študijski program: Računalništvo in informatika
Mentor: doc. dr. Peter Rogelj

Koper, september 2018

Ključna dokumentacijska informacija

Ime in PRIIMEK: Matej BRLEC

Naslov zaključne naloge: Mobilna aplikacija za odkrivanje škodljivih aditivov v živilskih izdelkih

Kraj: Koper

Leto: 2018

Število listov: 64	Število slik: 23	Število tabel: 11
Število prilog: 1	Število strani prilog: 1	Število referenc: 14

Mentor: doc. dr. Peter Rogelj

Ključne besede: mobilna aplikacija, programsko inženirstvo, Android, Java, OCR, Tesseract, SQLite, aditivi, zdravje

Izvleček:
Zaključna naloga predstavlja proces programskega inženirstva skozi uporaben praktični primer mobilne aplikacije za odkrivanje škodljivih aditivov v živilih. Dokument pokriva različne faze programskega inženirstva, in sicer, definicijo problema, študijo izvedljivosti, definicijo zahtev, načrtovanje, izvedbo in testiranje. Zadnji dve fazi, predaja in vzdrževanje, sta izpuščeni, saj gre za projekt zaključne naloge, kar pomeni, da se programski proces zaključi s končano zaključno nalogo. Uporaba formalnih faz programskega inženirstva lahko zagotovi programski izdelek višje kakovosti. Izdelan program je mobilna aplikacija za Android naprave, napisana v Java programskem jeziku. Optično prepoznavanje znakov (OCR) igra pomembno vlogo v tej aplikaciji in je implementirano z Tesseract pogonom za optično prepoznavanje znakov. Aplikacija omogoča uporabnikom zajemati fotografije sestavin na embalažah živilskih izdelkov, nad katerimi se izvede OCR postopek. Nato so sestavine analizirane in rezultat prikazan.

Key words documentation

Name and SURNAME: Matej BRLEC

Title of final project paper: Mobile application for detecting harmful food additives in food products

Place: Koper

Year: 2018

Number of pages: 64 Number of figures: 23 Number of tables: 11

Number of appendices: 1 Number of appendix pages: 1 Number of references: 14

Mentor: Assist. Prof. Peter Rogelj, PhD

Keywords: mobile application, software engineering, Android, Java, OCR, Tesseract, SQLite, food additives, health

Abstract:
This thesis seeks to demonstrate the process of software engineering through a practical and useful example of a mobile application for detecting harmful food additives in food products. The document covers the various software engineering phases, namely problem definition, feasibility study, requirements analysis, software design, implementation and testing. The last two phases of installation and maintenance are not covered due to the fact that this is a thesis project and therefore the product is presented as is. Following the formal phases of software engineering ensures a higher quality product. The developed software is a mobile application for Android devices written in the Java programming language. Optical character recognition (OCR for short) plays a central role in this application and it is implemented with the Tesseract OCR engine. The application allows users to take pictures of food ingredients from food packaging, on which the OCR process is performed and finally the ingredients are analyzed and the results displayed.

Zahvala

Zahvalil bi se rad vsem, ki so me na študijski poti podpirali, spodbujali ter mi stali ob strani. Zahvala gre tudi vsem profesorjem in asistentom, ki so mi predali potrebna znanja za pripravo te zaključne naloge. Prav tako izrekam zahvalo celotnem kolektivu UP FAMNIT, ki so moj študij sploh omogočili.

Še posebej bi se rad zahvalil mentorju doc. dr. Petru Roglju za naklonjen čas, podporo, kvalitetno svetovanje ter usmerjanje pri procesu priprave zaključne naloge.

Kazalo vsebine

Kazalo tabel

Kazalo slik

Kazalo prilog

Seznam kratic

ipd.	in podobno
t. j.	to je
npr.	na primer
oz.	oziroma
angl.	angleško
sl.	slovensko
OCR	optično prepoznavanje znakov (angl. optical character recognition)
t. i.	tako imenovani
itn.	in tako naprej

1 Uvod

Živimo v času, v katerem je pridobitev raznoraznih dobrin relativno enostavna. Velja celo, da za vsak tip izdelka obstaja več različnih proizvajalcev oz. ponudnikov. Velikokrat so ti proizvajalci globalni ter prodajajo izdelke na več trgih oz. državah. Med te dobrine seveda sodijo tudi živilski izdelki. Ti so še toliko bolj občutljivi, saj imajo v primerjavi z večino drugih dobrin v osnovi relativno kratek rok uporabe. Ne nazadnje pri prodaji živil rok uporabe ni vse. Kupce namreč bolj privlačijo npr. lep izgled, okusnost ter dišečnost. Da bi doesgli te lastnosti, se živilskim izdelkom dodaja razne prehranske dodatke, katerim pravimo tudi *aditivi*. To so razni konzervansi, zgoščevalci, razredčevalci, barvila, arome ipd [1]. Aditivi ter ostale prisotne sestavine morajo biti navedeni na embalaži živila. Rezultati raznih raziskav pravijo, da so lahko aditivi za zdravje potencialno nevarni ali celo škodljivi. Z dokazovanjem pravilnosti teh raziskav se sicer ne bom ukvarjal, saj to ni namen te zaključne naloge.

Namesto tega bom pripravil oz. razvil programsko rešitev, ki bo pomagala uporabnikom prepoznavati aditive na embalažah živil in jim ponudila informacije o morebitnih škodljivih učinkih prepoznanih aditivov. Rešitev bo predstavljena v obliki aplikacije za pametne mobilne telefone z operacijskim sistemom Android. Aplikacija bo s pomočjo optičnega prepoznavanja znakov (angl. krajše OCR) prepoznala navedene, potencialno škodljive, aditive ter uporabniku ponudila informacije o njih. Namen aplikacije bo torej informiranje uporabnikov. Poleg programske rešitve bo del zaključne naloge tudi pripadajoča dokumentacija. Razvoj bo namreč formalne narave, t. j. sledil bo programskemu procesu z jasno definiranimi koraki razvoja.

Namen naloge je širjenje zavesti o potencialno škodljivih snoveh v živilskih izdelkih in tudi učenje na tem praktičnem primeru. Na grobo lahko nalogo razdelimo na dva dela – dokumentiranje ter izvajanje (programiranje). Na strani dokumentiranja bom podrobneje spoznal ter tudi uporabil razne procese programskega inženirstva, ki so še posebej pomembni pri razvoju takšnih in drugačnih programskih rešitev. Na strani izvedbe bom utrdil uporabo meni že znanih tehnologij in osvojil nekatere nove tehnologije. Predvidevam, na primer, spoznanje z zgoraj omenjeno tehnologijo optičnega prepoznavanja znakov, Git sistemom, kjer bom hranil kodo, in tudi s samim dokumentiranjem kode s pomočjo JavaDoc. Menim, da mi bodo pridobljena znanja in izkušnje koristili pri nadaljnjem učenju ter morebitnih projektih.

2 Definicija problema

Dandanes je praktično nemogoče najti živilo brez aditivov. To še posebej velja za predelana živila. Živilske aditive se uporablja zaradi več razlogov, kot so preprečevanje kvarjenja živila (konzervansi), spreminjanje barve živila (barvila), poudarjanje okusa (ojačevalci okusa) ipd. Težava je v tem, da so nekateri aditivi lahko zdravju škodljivi. Vsi škodljivi aditivi niso nujno prepovedani, saj manjše količine načeloma niso (tako) škodljive ter so posledično dovoljene [2]. Proizvajalci živil morajo na embalaži izdelka seveda navesti vse uporabljene sestavine, vključno z aditivi. V večini primerov to za povprečnega potrošnika žal ni dovolj. Nespametno je namreč pričakovati, da bo vsak potrošnik sposoben prepoznavanja vseh aditivov, ki so navedeni s strokovnimi imeni in oznakami ter večkrat predstavljajo razne kemijske spojine. Vsekakor je nemogoče pričakovati, da bo potrošnik poleg vseh imen in oznak tudi poznal njihove morebiti zdravju škodljive, morebiti ugodne učinke. Skratka, potrošnikom bi prav prišlo posedovanje več informacij o hrani, ki jo uživajo, saj ta ne nazadnje vpliva na njihovo zdravje. Prepričati proizvajalce živil v boljše informiranje potrošnikov se ravno ne sliši kot najbolj učinkovita ideja.

Zato vidim rešitev v mobilni aplikaciji. Z mobilno platformo dosežemo veliko skupino potencialnih uporabnikov, kar je ena pomembnejših metrik za rešitve, ki stremijo k informiranju javnosti. Prednost mobilne platforme je tudi preverjanje živil neposredno v trgovini ali kjerkoli smo. Uporabnik bi z mobilno aplikacijo namreč fotografiral sestavine na embalaži izdelka. Sledila bi samodejna analiza fotografije oz. sestavin na fotografiji. Prepoznani aditivi bi bili primerno označeni oz. opisani glede na škodljivost. Rešitev torej dobesedno prinese potrebno znanje v potrošnikove roke.

3 Študija izvedljivosti

V študiji izvedljivosti preučimo, ali projekt lahko uspešno izpeljemo do konca. Morda manj opazen korak pri manjših, manj pomembnih projektih, kot je ta, vendar zelo pomemben in obsežen korak pri večjih oz. pomembnejših projektih. S študijo izvedljivosti namreč ugotovimo, ali smo sploh zmožni projekt realizirati, kot smo si ga zamislili. Projekt analiziramo z več plati, med katerimi najbolj izstopajo tehnična, operativna, ekonomska ter organizacijsko-socialna plat. Tako ugotovimo, ali obstajajo tehnologije, ki jih potrebujemo, ali je projekt operativno izvedljiv (zakonitost, kulturna sprejemljivost itn.), ali smo ekonomsko zmožni realizirati projekt, ali imamo na voljo dovolj časa za realizacijo projekta ipd. Če je rezultat študije izvedljivosti pozitiven, je to že zadosten razlog za pričetek z razvojem programskega izdelka.

Kot sem omenil, je zastavljen projekt relativno majhen ter se zato ne bom ubadal z veliko podrobnostmi, ki bi bile sicer prisotne pri večjih projektih. Tako bom na primer izpustil pravno ter kulturno izvedljivost, razne ocene tveganja, alternative, potrebno opremo ipd.

3.1 Tehnična izvedljivost

Mobilna aplikacija je razdeljena na več primarnih komponent, in sicer:

- zajemanje fotografij ter njihovo obrezovanje,
- razpoznavanje besedila v fotografiji,
- prepoznavanje besed v razpoznanem besedilu,
- popravljanje neprepoznanih besed ter
- označevanje prepoznanih besed na fotografiji.

Aplikacija potrebuje fotografijo deklaracije živila. Smiselno bi bilo pričakovati, da uporabnik lahko zajame fotografijo sestavin živila neposredno s pomočjo aplikacije. Za to bom uporabili privzeto aplikacijo za zajemanje fotografij na uporabniškem mobilnem telefonu. Operacijski sistem Android namreč omogoča uporabo oz. integracijo ostalih aplikacij znotraj naše aplikacije. To je realizirano tako, da uporabnik načeloma sploh

ne opazi, da je dejansko pričel uporabljati drugo aplikacijo. Razvoj novega modula za zajemanje fotografij bi bil potencialno časovno zelo potraten in neutemeljen. Potrebujem tudi možnost obrezovanja fotografij, saj si ne želimo nepotrebnega besedila v fotografiji (embalaže živil imajo poleg sestavin navedene še razne druge informacije in tudi sestavine v drugih jezikih). Besedilo, ki nas ne zanima, bi tudi upočasnilo proces razpoznavanja besedila iz fotografije. Za obrezovanje fotografij bom uporabil zunanjo knjižnico Android Image Cropper [3]. Posledično nas tehnična izvedljivost dotičnega modula ne skrbi, saj je zagotovljena s strani njegovih razvijalcev. Možnost implementacije knjižnice v aplikacijo je po eni strani tudi zagotovljena, saj projekt ustreza licenčnim in tudi sistemskih zahtevam knjižnice.

Razpoznavanje besedila iz fotografije bo prav tako implementirano s pomočjo zunanje knjižnice. Uporabil bom knjižnico imenovano tess-two [4]. ess-two je knjižnica, ki omogoča uporabo Tesseract OCR pogona na Android sistemu. Tesseract je najverjetneje najbolj dovršeno in uporabljano orodje za optično prepoznavanje znakov. Njegov razvoj se je začel pred približno tridesetimi leti v podjetju Hewlett-Packard (HP). Leta 2006 je podjetje Google začelo podpirati projekt ter praktično prevzelo nadzor nad razvojem [5].

Relevantne besede, torej aditive, bom hranil v lokalni podatkovni bazi. Sistem, ki ga bom uporabil za upravljanje s podatkovno bazo, bo SQLite [6]. SQLite je verjetno najprimernejša rešitev za moje potrebe, saj je primarno namenjena prenosljivosti. To sicer pomeni manjši nabor funkcij, ampak precej manjšo velikost ter porabo sistemskih sredstev. Za uporabo na mobilnih sistemih je to še posebej velika prednost. Prav tako bo predlagana rešitev uporabljala le nekaj osnovnih funkcij, tako da naprednih, manjkajočih, funkcij ne bom pogrešal. Primerjavo razpoznanih besed iz fotografije s seznamom aditivov v podatkovni bazi je precej preprosto realizirati. Gre namreč za primerjavo nizov, kar je ena od osnovnih funkcij večine sodobnih programskih jezikov.

Besede, ki ne bodo prepoznane v prejšnjem koraku prepoznavanja besed, bom ponudil uporabniku za ročni vnos. Iz zajete fotografije bom izrezal neprepoznane besede ter ob njih postavil vnosna polja, kjer bo uporabnik lahko vnesel oz. prepisal besedo na fotografiji. Obrez posameznih besed je enostaven, če poznam njihove koordinate. Koordinate lahko dobim s pomočjo prej omenjene knjižnice tess-two.

značevanje prepoznanih besed bo implementirano z risanjem na fotografijo. Ta funkcionalnost je že implementirana v Android sistemu oz. v programskem jeziku Java. Večji problemi se tu tako ne bi smeli pripetiti. Morebiti bom prikazoval še dodatne informacije, za katere bi uporabil razne poglede operacijskega sistema Android (angl. view) [7]. Pogledi v Android sistemu so osnovne komponente za izdelavo grafičnega uporabniškega vmesnika.

Preostanejo le še navigacija oz. grafični uporabniški vmesnik, nastavitve, vmesna

komunikacija oz. povezanost med komponentami in podobne zadeve. To pa so vse reči, za katere še ni potrebno izgubljati besed. To so seveda osnovne komponente večine sodobnih programskih rešitev ter so tudi izvedljive.

Iz ugotovljenega sledi, da imam vsa potrebna znanja za razvoj izbrane aplikacije in da sem seznanjen s potrebnimi tehnologijami, ki jih bom uporabil v svoji aplikaciji. Zaključim torej lahko, da je razvoj s tehnične plati mogoč.

3.2 Operativna izvedljivost

Mobilna aplikacija oz. projekt na sploh ne krši nobenega zakona. Prav tako ne opažam morebitne moralne spornosti. Nasprotno, menim, da je eden od rezultatov projekta doprinos h kakovosti življenja posameznikov.

Kar se tiče zunanjih knjižnic, ki jih imam namen uporabiti, so vse licencirane pod Apache 2.0[1] licenco, ki je zelo dovoljujoča. Dovoljuje seveda tudi nadaljnjo distribucijo – kar mi je najbolj pomembno.

Težava se pojavi le pri zbirki aditivov. Primerno zbirko sem sicer našel, a nisem uspel priti v kontakt z avtorjem ter posledično nisem pridobil soglasja za uporabo le-te. Zaradi časovnih omejitev trenutno tudi nimam časa pripraviti svoje zbirke. Tako sem se odločil, da bom za potrebe priprave prototipa nekaj podatkov povzel po najdeni zbirki. Za vnaprej je predvidena priprava lastne zbirke aditivov, ki bo naknadno izšla v obliki posodobitev mobilne aplikacije. Za pripravo posodobitev bo skrbel skrbnik sistema.

3.3 Ekonomska izvedljivost

Znano mi je, da bodo finančni stroški razvoja praktično ničelni oz. zanemarljivo majhni. Prav tako projekt ni namenjen prinašanju dobička. Posledično lahko sklenem, da je projekt kot tak finančno izvedljiv, kar mi je bilo znano že pred opravljanjem študije izvedljivosti. Bo pa zato v projekt vloženega veliko dela, na kar sem tudi pripravljen. Posledično bi v tem primeru lahko izpustili korak ekonomske izvedljivosti.

Korak je bil dodan le zato, ker je običajno zelo pomemben pri razvoju programske opreme. Tako podjetja, ki se ukvarjajo z razvojem programskih rešitev, in tudi posamezniki navadno ciljajo na zaslužek. Brez zaslužka namreč ni mogoče nadaljevati s poslovanjem. Študija ekonomske izvedljivosti je namenjena prav temu. Upoštevamo vse stroške, ki jih bomo imeli z razvojem programske rešitve. Ti stroški lahko na primer vključujejo plačilo vpletenih oseb, stroške potrošnega materiala (papir, pisala,

[1] http://www.apache.org/licenses/LICENSE-2.0

optične medije ...), stroške, ki jih imamo z delovnim prostorom (ogrevanje, elektrika, najemnina, ...), nakup licenc za razna orodja ali tehnologije, logistiko, promocijo, patentiranje in še veliko več. Kaj hitro lahko ugotovimo, da je stroškov lahko ogromno ter bi na mnoge od njih lahko pozabili ali jih podcenili, če ne bi opravili temeljite študije. Poleg stroškov moramo seveda tudi oceniti, koliko bi lahko zaslužili z našim izdelkom. Če se preračunani stroški in zaslužek ne izidejo to pomeni negativen, ničeln ali nezadostni dobiček. V tem primeru projekt načeloma opustimo. To seveda vedno ne drži v primerih, ko dobiček ni glavna prioriteta.

3.4 Organizacijsko-socialna izvedljivost

Glede na to, da na projektu delam sam, s pomočjo mentorja, bom razdelek namenil časovni izvedljivosti.

Diplomsko nalogo ter posledično proces programskega inženirstva nameravam zaključiti v letošnjem študijskem letu. To pomeni, da imam od tega trenutka na voljo še nekaj več kot tri mesece. Diplomska naloga mora namreč biti oddana do 10. avgusta. V tem času moram imeti pripravljen dokončan prototip mobilne aplikacije ter pripadajočo dokumentacijo, strnjeno v zaključno nalogo.

Implementacija osnovne funkcionalnosti aplikacije ne bo predstavljala težav, saj večinoma vključuje razvoj komponent, s katerimi že imam izkušnje. Ocenim lahko, da bodo za njen razvoj potrebna dva tedna trdega dela, ki pa bosta v praksi najverjetneje razširjena na okoli štiri tedne bolj "sproščenega" dela. Pisanje dokumentacije prav tako ni samo po sebi kompleksno, a utegne biti včasih zamudno. Ocenjujem, da bo za potrebe dokumentiranja dovolj od tri do pet tednov. Dodal bi še en teden za razne popravke v dokumentaciji in tudi v aplikaciji ter testiranje. To skupno nanese največ deset tednov, kar mi dopušča še kakšen teden za odzive na nepredvidene dogodke.

Preostali čas (če bo na voljo) nameravam uporabiti za optimizacijo aplikacije, dodajanje morebitnih manjših izboljšav, katerih potrebo bi opazil med testiranjem, dodelavo grafičnega vmesnika ipd. Nekaj časa bo tudi potrebnega za posvete z mentorjem ter mentorjev pregled naloge. Za to bi rad zagotovil vsaj dva tedna.

3.5 Ugotovitev

Z uspešno opravljeno študijo izvedljivosti ugotovimo, da obstaja velika verjetnost za uspešno realizacijo zastavljenega projekta v celoti. Čeprav to ni naloga študije izvedljivosti, iz nje lahko še ugotovimo, na katerih področjih projekta lahko pride do komplikacij. Posledično lahko to znanje uporabimo v naslednjih fazah programskega

inženirstva. Če bi definirali več različic za razvoj projekta, bi vsako od njih analizirali ter izbrali najprimernejšo.

Ugotovitev moje študije izvedljivosti je, da je zastavljen projekt izvedljiv iz vseh analiziranih plati. To pomeni, da bom s projektom nadaljeval. Posledično pričakujem tudi pravočasno ter uspešno zaključen projekt.

4 Definicija zahtev

V fazi definicije (opredelitve) zahtev opišemo vse želene lastnosti našega programskega izdelka. Ne smemo pa pozabiti na opis okolja, kjer se bo programski izdelek uporabljal. S poznavanjem okolja (uporabniki, naprave ipd.) si lahko lažje predstavljamo realno uporabo našega izdelka ter tako izdelek prilagodimo okolju. To pa s seboj prinese pozitivnejšo uporabniško izkušnjo ter višjo kakovost programske rešitve.

Rezultat definicije zahtev je natančen in podroben dokument, imenovan specifikacija zahtev (angl. software requirements specification – SRS). Po končani opredelitvi zahtev sledi še njihova analiza. Z analizo skušamo preveriti popolnost, konsistentnost, smiselnost, ipd. specifikacije zahtev. Skratka, preverimo, ali smo predstavili vse želene funkcije izdelka, ali so opredeljene zahteve med seboj združljive ter ali je vse skupaj še vedno smiselno in sploh mogoče realizirati s sredstvi, ki jih imamo na voljo.

Če razvijamo programski izdelek po naročilu, je na tej stopnji še posebej pomembno sodelovanje z naročnikom. To je namreč dokument, ki se bo uporabljal v času razvoja – načrtovanje ter validacija (testiranje). Napake v tej fazi lahko ostanejo neopažene zelo dolgo, tudi do predaje izdelka naročniku. Odprava napak je draga, saj se moramo vrniti na vse faze, kjer so napake prisotne, ter jih odpraviti. Proces odpravljanja napak lahko privede tudi do novih napak, zato sta pozornost in natančnost zelo pomembni.

Specifikacija zahtev je praviloma zelo obsežna – vsebuje več poglavij. Zastavljen projekt je relativno enostaven, zato bom uporabil nekoliko okrnjeno oz. poenostavljeno različico omenjenega dokumenta. Osredotočil se bom na elemente, ki se mi zdijo pomembnejši ter posledično predstavljajo večje tveganje za nastop težav. S podrobnim opisom in analizo bom to tveganje nekoliko zmanjšal.

4.1 Splošni opis

Programski izdelek je aplikacija za pametne mobilne telefone z operacijskim sistemom Android. Izbrana rešitev ne predstavlja kompleksne aplikacije – primarno bi jo lahko razdelili na štiri glavne komponente. Po sekvenci uporabe naletimo najprej na zajemanje in urejanje fotografije, nato na razpoznavanje besedila iz fotografije, sledi pomensko prepoznavanje oz. popravljanje besedila ter na koncu še označitev oz. prikaz rezultata ali informacij.

Šibka točka je tukaj razpoznavanje besedila iz fotografije, kjer tudi pričakujem največ težav pri razvoju aplikacije. Težave bodo bolj ali manj nefunkcijske. Predvsem me skrbi nezanesljivost v določenih primerih. Nekaj izmed njih je omenjenih v naslednjem odstavku. Posledično sem se odločil, da nameravam z odpravo takšnih težav začeti v dobi optimizacije, ki bo nastopila, kot sem že omenil, po končanem razvoju osnovne različice aplikacije.

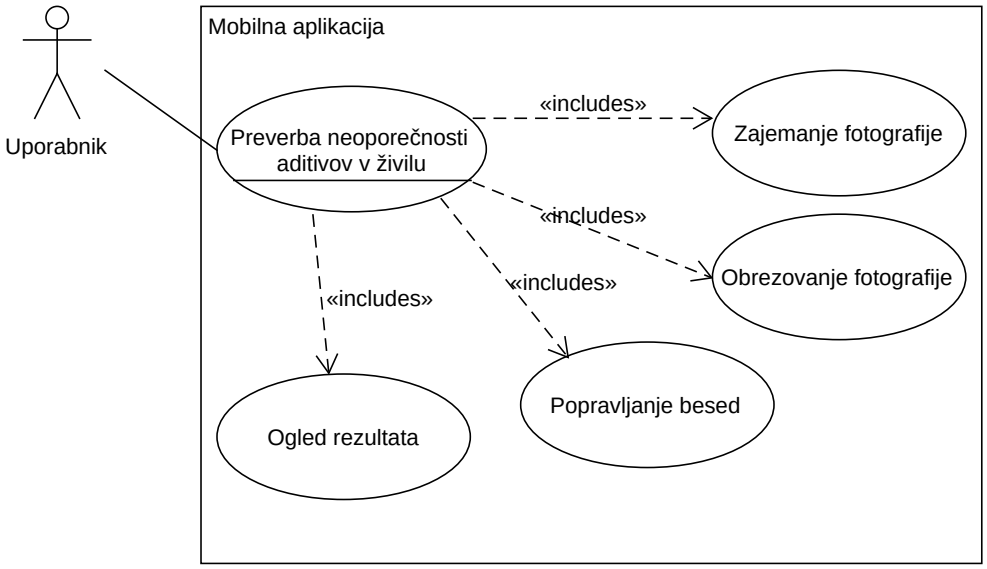
Zajemanju fotografij bom moral posvetiti tudi nekoliko več pozornosti, saj neposredno vpliva na kvaliteto razpoznavanja besedila iz fotografije. Vseeno se zavedam, da tu ne bom imel velike moči. Namreč, vplivati ne morem na veliko stvari, kot so na primer primerna barva ter oblika embalaže živila, kakovost uporabnikovega fotoaparata ter uporabnikovo sposobnost zajemanja kakovostnih fotografij. Zato se zavedam, da ne bo mogoče pripraviti stoodstotno popolnega izdelka. Verjamem pa, da bi lahko negativen učinek teh težav zmanjšal, vendar ne odpravil, z raznimi kompleksnimi rešitvami. Lahko bi na primer implementiral modul za manipulacijo s fotografijami, ki bi fotografije vnaprej obdelal v bolj primerno obliko. Žal pa bi ta pristop zahteval preveč dragocenega časa. Lahko bi to bila smiselna usmeritev za nadgradnjo izdelka v prihodnosti.

Ker sem že omenil uporabnike, velja tudi povedati kaj o ciljni skupini aplikacije. Aplikacija ima seveda specifičen namen. Ta je, kratko povedano, informiranje javnosti. Podrobneje bi lahko govorili tudi o ponujanju zelo prenosne baze (specifičnega) znanja in tudi o promoviranju zdravega načina življenja. Ciljna skupina so tako ljudje, ki si želijo zdravega življenjskega sloga. Seveda pa pričakujem tudi ostale tipe uporabnikov. Predstavljam si, da so to večinoma povprečni ljudje, ki niso nujno zelo vešči uporabe računalniških sistemov. Na podlagi tega velja sklepati, da bi bilo smiselno pripraviti aplikacijo, ki bi bila enostavna za uporabo. Iz poznanih podatkov kaj več ni mogoče sklepati. Pri konkretnejših projektih bi te podatke pridobili z raznimi študijami (anketiranje), upoštevanjem ustaljenih praks, posvetovanjem s strokovnjaki ipd. Za moj primer bo usmeritev v enostavnost, preglednost in odzivnost dovolj. Več o tem sledi v podpoglavju nefunkcijskih zahtev (4.3).

4.2 Funkcijske zahteve

Sledi predstava funkcijskih zahtev v obliki diagrama primera uporabe ter pripadajočega opisa.

4.2.1 Primer uporabe



Slika 1: Diagram primera uporabe aplikacije.

Diagram primera uporabe na sliki 1 prikazuje interakcijo med uporabnikom in mobilno aplikacijo. Sledijo podpoglavja s formalnimi opisi posameznih primerov uporabe. Najprej so predstavljeni primeri uporabe na nižjem nivoju, ki so vključeni v celosten primer uporabe preverbe neoporečnosti aditivov v živilu, ki je predstavljen zadnji.

Zajemanje fotografije

Tabela 1: Opis primera uporabe zajemanja fotografije.

Naziv	Zajemanje fotografije
Akterji	Uporabnik
Zahteve	Aplikacija mora uporabniku nuditi možnost preverjanja neoporečnosti aditivov v živilu.
Prožilec	Uporabnik zažene aplikacijo ter pritisne na gumb za zajemanje fotografij.
Osnovni potek	1. Uporabnik usmeri fotoaparat mobilnega telefona proti sestavinam na embalaži živila. 2. Uporabnik pritisne gumb za zajem slike. 3. Uporabnik potrdi zajeto fotografijo.
Stanje po zaključku	Aplikacija hrani zajeto fotografijo deklaracije živila.
Alternativni poteki	3a: Uporabnik ni zadovoljen z zajeto fotografijo: 3a1. Uporabnik zavrne zajeto fotografijo ter se vrne na 1. korak. 3b: Uporabnik si premisli in ne zajame fotografije: 3b1. Uporabnik konča primer uporabe.

Obrezovanje fotografije

Tabela 2: Opis primera uporabe obrezovanja fotografije.

Naziv	Obrezovanje fotografije
Akterji	Uporabnik
Zahteve	Uspešno zajeta fotografija.
Prožilec	Uporabnik je uspešno zajel fotografijo.
Osnovni potek	1. Uporabnik spreminja velikost pravokotnika za obrez fotografije. 2. Uporabnik potrdi obrez fotografije.
Stanje po zaključku	Aplikacija hrani obrezano fotografijo ter prične z razpoznavanjem besedila v fotografiji.
Alternativni poteki	1a: Uporabnik ne želi obrezati fotografije: 1a1. Uporabnik konča primer uporabe.

Ogled rezultata

Tabela 3: Opis primera uporabe ogleda rezultata.

Naziv	Ogled rezultata
Akterji	Uporabnik
Zahteve	Uspešno izveden postopek optičnega prepoznavanja besedila.
Prožilec	Uporabnik pritisne na razpoznano besedo.
Osnovni potek	1. Uporabnik si ogleda podatke o aditivu, če je razpoznana beseda aditiv. 2. Uporabnik zapre pogled podrobnosti razpoznane besede. 3. Uporabnik pritisne gumb za zaključek
Stanje po zaključku	Uporabnik ugotovi (ne)oporečnost aditivov v izbranem živilu. Aplikacija se povrne v prvotno stanje – začetni zaslon.
Alternativni poteki	1a: Uporabnik si ne ogleda rezultata: 1a1. Uporabnik konča primer uporabe. 2a: Uporabnik ne pritisne na gumb za zaključek: 2a1. Uporabnik konča primer uporabe.

Popravljanje besed

Tabela 4: Opis primera uporabe popravljanja neprepoznanih besed.

Naziv	Popravljanje besed
Akterji	Uporabnik
Zahteve	Možnost ročnega popravljanja nepravilno razpoznanih besed, njihovo združevanje in razdruževanje v imena aditivov.
Prožilec	Uporabnik pritisne na neprepoznano (neobarvano) besedo.
Osnovni potek	1. Uporabnik prepiše nepravilno prepoznano besedo s pravilno. 2. Uporabnik potrdi popravek besede.
Stanje po zaključku	Aplikacija hrani pravilnejše stanje razpoznanih besed.
Alternativni poteki	1a: Uporabnik prepiše nepravilno prepoznano besedo: 1a1. Uporabnik ne potrdi popravka. 1a2. Uporabnik konča primer uporabe. 1b: Uporabnik združi trenutno izbrano besedo z eno od sosednjih: 1b2. Uporabnik konča primer uporabe. 1c: Uporabnik razdeli besedo na več delov: 1c2. Uporabnik konča primer uporabe. 1d: Uporabnik ne prepiše nepravilno razpoznane besede: 1d2. Uporabnik konča primer uporabe.
Razno	Če uporabnik ne želi oz. pomanjkljivo opravi ta korak, obstaja večja verjetnost za nepopoln rezultat.

Preverba neoporečnosti aditivov v živilu

Tabela 5: Opis primera uporabe pregleda neoporečnosti aditivov v živilu.

Naziv	Preverba neoporečnosti aditivov v živilu
Akterji	Uporabnik
Zahteve	Uporabnik želi preveriti neoporečnost aditivov v živilu.
Prožilec	Uporabnik zažene aplikacijo ter pritisne na gumb za zajemanje fotografij.
Osnovni potek	1. Uporabnik zajame fotografijo. 2. Uporabnik obreže fotografijo. 3. Uporabnik popravi nerazpoznane besede. 4. Uporabnik si ogleda rezultat.
Stanje po zaključku	Uporabnik ugotovi (ne)oporečnost aditivov v živilu, aplikacija se povrne v začetno stanje.
Alternativni poteki	1a: Uporabnik ne želi zajeti fotografije: 1a1. Uporabnik konča primer uporabe. 2a: Uporabnik ne želi obrezati fotografije: 2a1. Uporabnik zapre aplikacijo. 3a: Uporabnik pomanjkjivo ali sploh ne določi pomena neprepoznanim besedam: 3a1. Nadaljuje na naslednji korak. 3a2. Uporabnik konča primer uporabe. 4a: Uporabnik si ogleda morebiti nepopoln rezultat v kolikor se je zgodili 3a: 4a1. Uporabnik konča z ogledom. Aplikacija se povrne v prvotno stanje. 4a2. Uporabnik konča primer uporabe.

4.3 Nefunkcijske zahteve

Za programsko rešitev ni dovolj, da deluje pravilno. Potrebne so tudi marsikatere ostale lastnosti. O nefunkcijskih lastnostih govorimo takrat, ko naletimo npr. na preprostost, dostopnost, stabilnost, prepustnost, odzivnost ipd. Na enak način, kot ima vsak projekt različne funkcijske zahteve, ima tudi različne nefunkcijske zahteve. V tem projektu so nefunkcijske zahteve pretežno odvisne oz. se nanašajo na uporabnika. Kot sem že omenil na koncu podpoglavja 4.1, bo nefunkcijska usmeritev pretežno potekala v smeri enostavnosti, preglednosti in odzivnosti.

- **Enostavnost** bom na eni strani dosegel z linearnostjo aplikacije, kar pomeni direktne prehode od začetka do konca. Drugače rečeno, napredovanje v aplikaciji je možno pretežno le v eno smer (naprej). Na drugi strani bom enostavnost tudi dosegel s preprostim grafičnim vmesnikom (jasen ter preprost brez nepotrebnih elementov). Stopnjo dosežene enostavnosti lahko preverim s spremljanjem prvega srečanja testnih uporabnikov s pripravljeno aplikacijo in tudi z njihovimi povratnimi informacijami.

- **Preglednost** bo delno zagotovljena z zgoraj omenjeno preprostostjo. Grafični vmesnik igra veliko vlogo v tem segmentu. Preglednost bo še posebej pomembna pri končnem prikazovanju rezultata – označevanje aditivov na zajeti fotografiji, kjer bo uporaba pravilnih barv, oblik in stilov ključna. Stopnjo dosežene preglednosti bom preveril predvsem s povratnimi informacijami testne skupine uporabnikov.

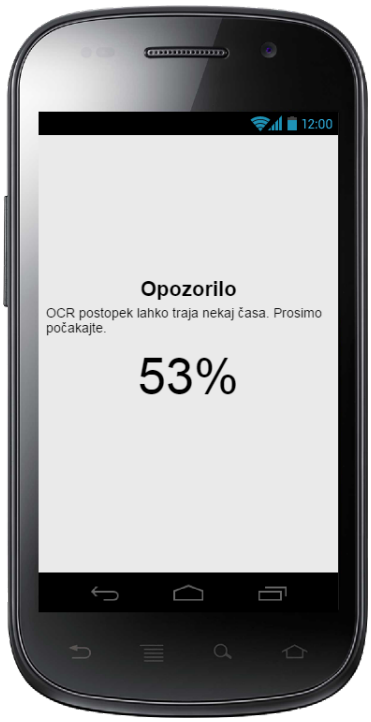
- **Odzivnost** bom zagotovil z enostavnostjo aplikacije, kjer enostavnost cilja na preprosto in hitro programsko kodo. To pomeni pametno uporabo raznih metod in modulov v fazi načrtovanja oz. kasneje izvedbe. Izjema je seveda kompleksna knjižnica za optično prepoznavanje znakov. Njeno časovno zahtevnost bom deloma omejil z modulom za obrezovanje fotografij. Z manjšimi vhodnimi podatki (fotografijo) namreč v povprečju dosežemo hitrejšo izvedbo obdelave. Odzivnost aplikacije bom prav tako preveril s povratnimi informacijami testne skupine uporabnikov. Če bom imel dostop do več različnih naprav, bom tudi sam lahko do neke mere preveril odzivnost.

- **Dostopnost** aplikacije igra ključno vlogo pri doseganju cilja projekta, kateri je, naj spomnim, ozaveščanje potrošnikov o potencialno škodljivih živilskih aditivih. Menim, da bo brezplačnost aplikacije igrala veliko vlogo pri povečanju dostopnosti. Obstaja tudi možnost objave aplikacije v raznih t. i. trgovinah z aplikacijami (Google Play, F-Droid). Doseženo stopnjo uporabe bi lahko preveril s spremljanjem števila prenosov.

4.4 Uporabniški vmesnik

Predviden grafični vmesnik pomembnejših pogledov aplikacije (ogled rezultata) je prikazan na slikah 4 ter 5. Poleg teh dveh pogledov sta tu še začetni zaslon ter pogled, ki naznanja napredek procesa razpoznave besedila. Začetni zaslon predstavlja preprost pogled z nekaj informacijami ter gumbom za prižig kamere (slika 2), medtem ko pogled napredka le prikazuje napredek OCR procesa (slika 3).

Slika 2: Predviden izgled začetnega zaslona.



Slika 3: Predviden izgled prikaza napredka razpoznave.

Pogled rezultata (slika 4) se začne z naslovom aktivnosti. Sledijo mu krajša uporabniška navodila za pregled rezultata. Sledi obrezana fotografija z ustrezno označenimi aditivi. Razvidnih je nekaj različno obarvanih besed. Barve označujejo stopnjo nevarnosti prepoznanih aditivov. Rdeča pomeni zelo nevarno, oranžna nevarno, zelena nenevarno, siva sumljivo, sinje modra neznano, medtem ko neobarvano pomeni, da beseda bodisi ni aditiv ali ni bila pravilno razpoznana. Ob pritisku na poljubno besedo se pojavi nov pogled s podrobnostmi izbrane besede. Če je izbrana beseda aditiv, bo pogled opremljen s podrobnostmi o aditivu. Predviden izgled tega pogleda je prikazan na sliki 5. Pogled se začne z vrstico za združevanje besed (funkcionalnost le-te je podrobneje opisana v podpoglavju 5.3.8 – *Popravljanje besed*). Sledi vnosno polje, v katerem je napisana razpoznana beseda. Če je to aditiv oz. del aditiva, bo ozadje zaslona obarvano skladno s stopnjo nevarnosti dotičnega aditiva. Prav tako bo prisoten segment s podrobnostmi o aditivu, ki vključuje celotno ime aditiva, E število ter morebiten opis

Slika 4: Predviden izgled prikaza rezultata.

Slika 5: Predviden izgled prikaza podrobnosti izbranega aditiva.

aditiva. "Drsenje" (angl. swype) levo in desno omogoča premik med besedami. Vrnitev na prejšnji pogled prepoznanih aditivov sprožimo s pritiskom na Androidov gumb "nazaj", tj. navadno trikotnik. Ogled celotnega rezultata zaključimo s pritiskom na tipko "KONČAJ", kar sproži ponoven zagon aplikacije oz. vrnitev v začetni položaj. Namen tega je hitra možnost preučitve škodljivosti naslednjega živilskega izdelka.

4.5 Ostale zahteve

Za namestitev aplikacije uporabnik potrebuje napravo z operacijskim sistemom Android različice 6 (*Marshmallow*) ali novejšim. Poleg tega mora naprava vsebovati tudi fotoaparat ter aplikacijo za zajemanje fotografij. Kvaliteta fotografij ter posledično kvaliteta fotoaparata in sposobnost zajemanja kvalitetnih fotografij igrajo velik pomen v aplikaciji, saj vplivajo neposredno na proces razpoznavanja besed (aditivov) iz fotografije.

5 Načrtovanje

Proces načrtovanja programskega izdelka je najlažje opisati z izjavo, da je to pretvorba definicije zahtev, katere rezultat je specifikacija zahtev, v načrtovalski pogled sistema. Pri načrtovanju izhajamo iz funkcijskih in nefunkcijskih zahtev dotičnega problema ter vztrajno snujemo načrt njegove rešitve.

Načrtovanje navadno razdelimo na dve fazi – načrtovanje sistema ter načrtovanje komponent. Načrtovanje sistema je namenjeno določitvi arhitekture programskega sistema. To pomeni smiselno razdelitev programskega izdelka na več manjših delov oz. komponent. Vsako komponento moramo dobro definirati – predstaviti njene lastnosti, zahteve, odvisnosti ipd. Rezultat je dobro definiran sistem manjših komponent, ki se še bolj podrobno načrtuje pri fazi načrtovanja komponent. Razlog za dekompozicijo problema na komponente tiči v tem, da je obravnavanje posamičnih komponent mnogo enostavnejše od obravnavanja celotnega, kompleksnega sistema. Posledično tudi prinaša manjše možnosti za napake [10].

Zaradi neobsežnosti projekta sem se odločil, da bom fazi združil ter obravnaval skupno v poglavju načrtovanja.

5.1 Načrtovalski cilji

Načrtovalske cilje določimo pretežno iz naših nefunkcijskih zahtev. Velikokrat tudi izberemo splošno priznane pozitivne lastnosti programskih produktov, kot so varnost, zanesljivost, učinkovitost, razširljivost ipd. Potrebno je tudi omeniti, da so si cilji lahko nasprotujoči, zato se običajno usmerimo v le nekaj od njih [10]. Z definicijo načrtovalskih ciljev tako določimo, čemu bomo namenili prioriteto pri načrtovanju. Če se na primer osredotočimo na varnost, bomo med načrtovanjem še posebej previdni glede možnih ranljivosti sistema.

Moji načrtovalski cilji bodo moje nefunkcijske zahteve razen zahteve dostopnosti. Nanjo namreč skoraj nimam vpliva med fazama načrtovanja in izvedbe. Tako se bom osredotočil na čim bolj uporabniku enostavno, pregledno in odzivno rešitev. Z načrtovano enostavnostjo nameravam tudi doseči robustno aplikacijo, saj uporabniki ne bodo imeli veliko možnosti za povzročanje napak z nepravilno uporabo le-te. Kot cilj bi morda dodal še razširljivost, saj bo zaključen izdelek tega procesa le osnovna oblika

rešitve. To pa morebiti pomeni pomanjkanje nekaterih zelo koristnih funkcionalnosti.

5.2 Dekompozicija programskega sistema

Faza je namenjena dekompoziciji programskega sistema na več manjših komponent. Predlagan programski izdelek ne predstavlja kompleksnega programskega sistema, zato bo delitev na manjše komponente minimalna. Programski sistem sestavlja le mobilna aplikacija, katere komponente so med seboj tesno povezane.

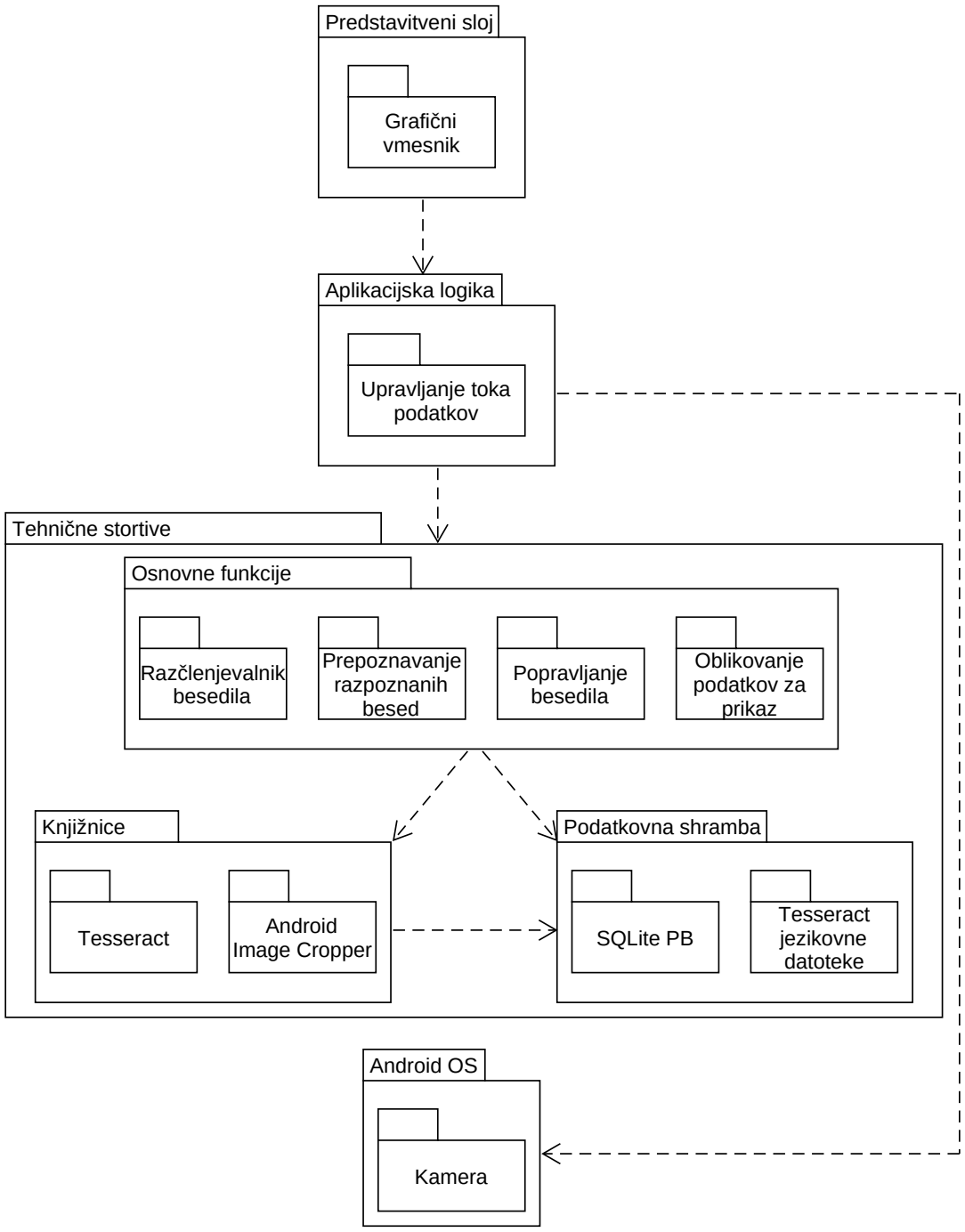
5.2.1 Logična arhitektura mobilne aplikacije

Logična arhitektura razdeli sistem, v tem primeru mobilno aplikacijo, na logično razdeljene skupine. Te skupine običajno organiziramo v sloje oz. podobne logične strukture. Za sloje je značilno, da so zgornji sloji navadno odvisni od spodnjih, tj. komponente zgornjih slojev uporabljajo storitve komponent spodnjih slojev [8].

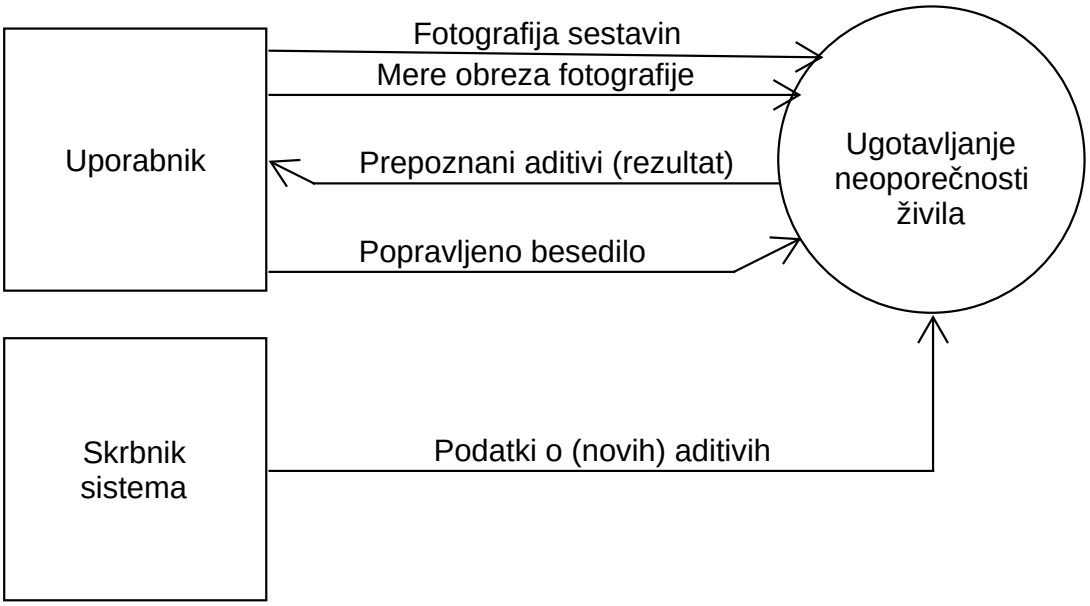
Mobilno aplikacijo sem razdelil na tri sloje. V paketnem diagramu (slika 6) je prisoten tudi četrti sloj, ki sicer predstavlja Android operacijski sistem, oz. napravo z omenjenim sistemom. Razlog za vključitev tega sloja tiči v tem, da je kamera naprave ponujena s strani operacijskega sistema, torej zunaj mojega vpliva. Lahko bi jo umestil v tretji sloj, h knjižnicam, ampak menim, da v svojem, najnižjem sloju natančneje predstavlja dejansko arhitekturo.

Ostali sloji so še predstavitveni sloj, aplikacijska logika ter tehnične storitve. Ti trije sloji so hkrati tudi najbolj tipični sloji logičnih arhitektur objektno usmerjenih[1] projektov [8]. Predstavitveni sloj običajno opravlja vlogo navigacije in prikaza informacij. Interakcije so primerno obdelane v sloju aplikacijske logike s pomočjo raznih rokovalnikov. Kot je razvidno iz paketnega diagrama (slika 6), moja aplikacijska logika predstavlja bolj ali manj le nalogo upravljanja toka podatkov (ponazorjen z diagrami toka podatkov na slikah 7, 8 ter 9). Razlog za to tiči v tem, da bom pri izdelavi aplikacije uporabil nekaj že obstoječih knjižnic in modulov. Manjkajoče module bom razvil sam. Tako je skoraj vse dogajanje prisotno v sloju tehničnih storitev.

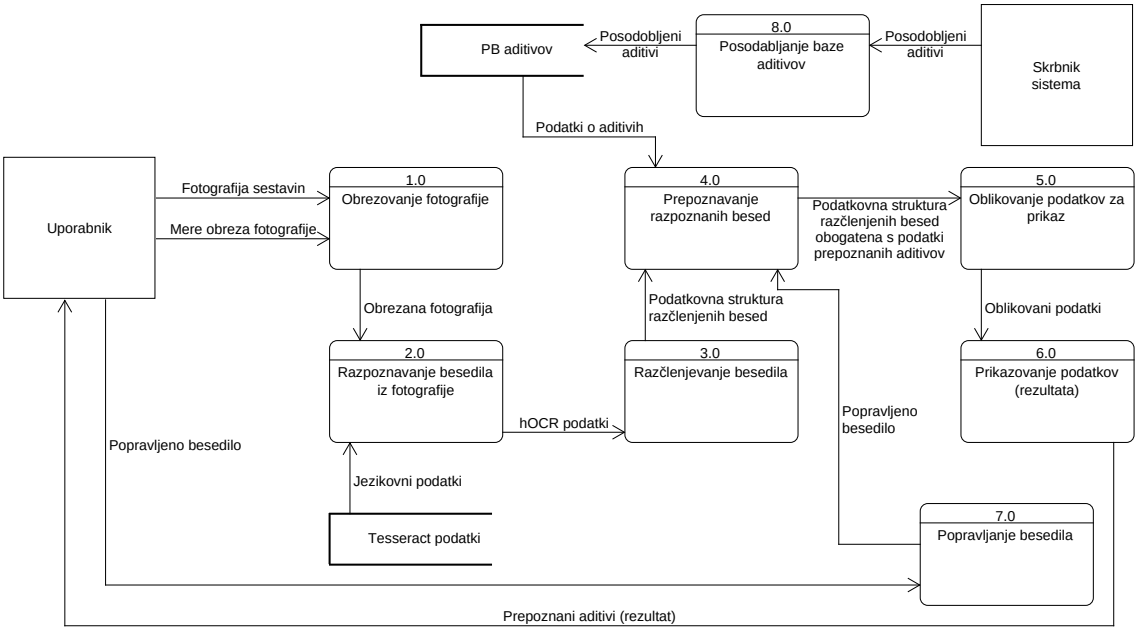
[1] Objektna usmeritev prikazuje realne entitete v objektni obliki



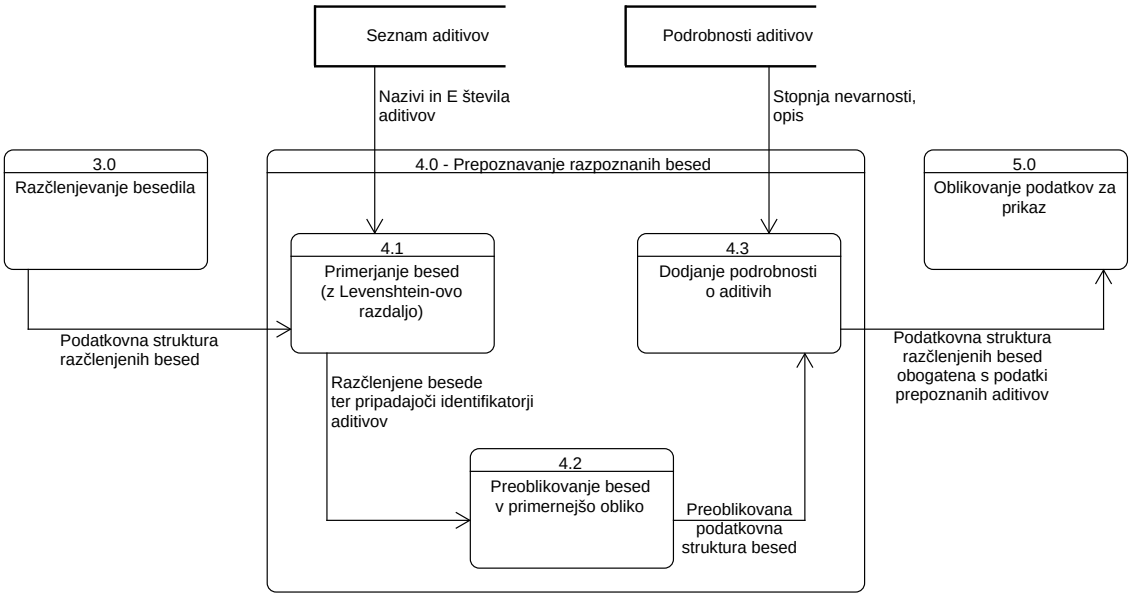
Slika 6: Paketni diagram mobilne aplikacije.



Slika 7: Diagram toka podatkov mobilne aplikacije na nivoju 0.



Slika 8: Diagram toka podatkov mobilne aplikacije na nivoju 1.



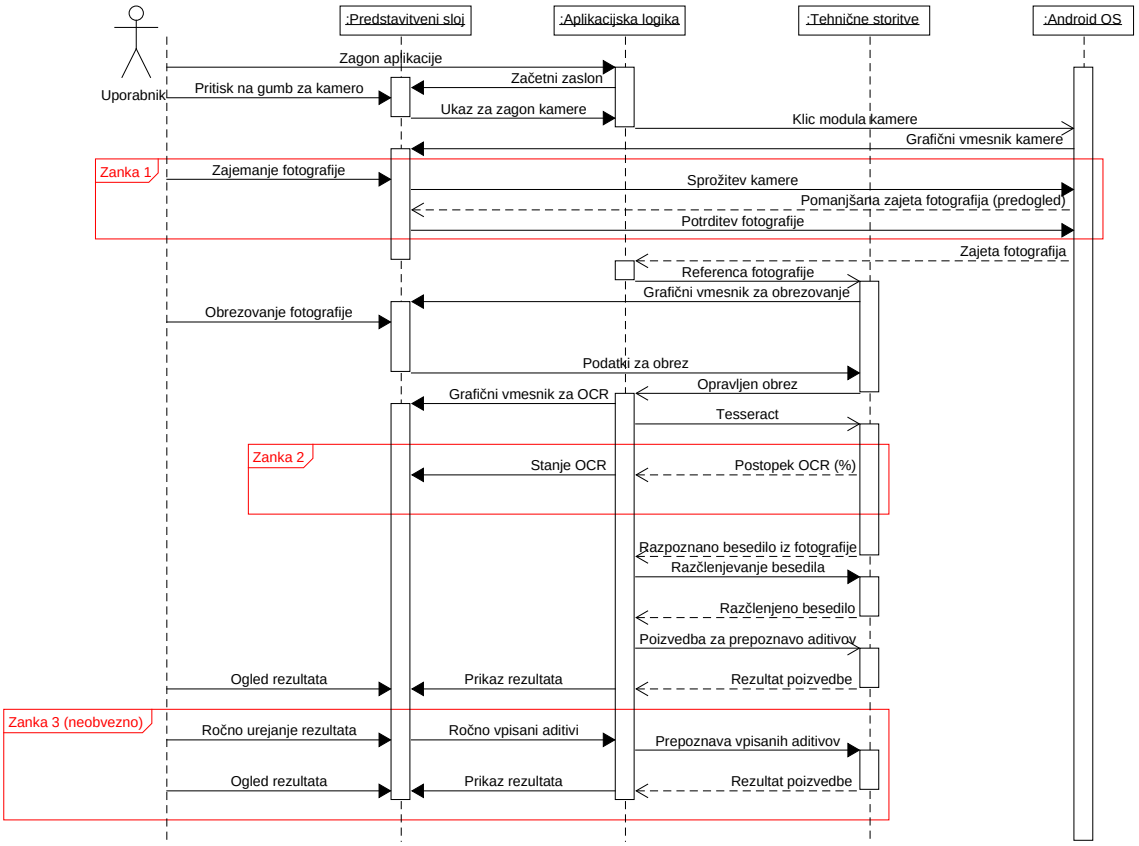
Slika 9: Diagram toka podatkov mobilne aplikacije na nivoju 2. Diagram podrobneje ponazarja 4. korak iz prejšnjega nivoja, t. j. prepoznavanje razpoznanih besed.

Vhod procesa prepoznave razpoznanih besed (4.0) je preprosta struktura podatkov o posamezni razpoznani besedi (niz s presledkom ločenih podatkov). V procesu 4.1 se razbrano besedo primerja z aditivi v podatkovni shrambi. Primerjava poteka s pomočjo Levenshteinove razdalje, katero bom uporabil kot merilo odstopanj med dvema besedama. Tako lahko prezrem manjša odstopanja in optimistično nekoliko izboljšam prepoznavo besed.

Po primerjavi razpoznanih besed bodo te preoblikovane v primernejšo obliko – prikazano na razrednem diagramu (slika 12) v podpoglavju 5.3.4 Razčlenjevanje besedila. Sledi dodajanje podrobnosti aditivov besedam, ki so aditiv oz. del aditiva. Ta korak vključuje le poizvedbo na podatkovno shrambo podrobnosti aditivov po E številu aditiva ter zapis podrobnosti v podatkovno strukturo. Dopolnjena podatkovna struktura je nato posredovana procesu oblikovanja podatkov za prikaz.

Z druge plati lahko pogledamo še potek dogodkov, katerega sem prikazal s sekvenčnim diagramom (slika 10). Diagram prikazuje uporabnika ter prej omenjene štiri sloje. Cilj diagrama je predstaviti potek komunikacije med posameznimi sloji. Iz sekvenčnega diagrama lahko običajno tudi razberemo podatke o posameznih komponentah in tipih podatkov, ki se prenašajo med njimi. Posledično si že lahko predstavljamo neke osnovne programske strukture teh komponent, kar bo nedvomno v pomoč pri podrobnejšem načrtovanju.

Iz diagrama lahko razberemo, da se ob zagonu mobilne aplikacije najprej prikaže



Slika 10: Sekvenčni diagram mobilne aplikacije.

začetni zaslon. Nato lahko uporabnik zažene kamero operacijskega sistema s pritiskom na ustrezen gumb. Kamera se inicializira in prikaže pripadajoči uporabniški vmesnik na zaslonu. Uporabnik lahko sedaj zajame fotografijo embalaže z deklaracijo živila. Če uporabnik oceni, da fotografija ni primerna (nizka kakovost, pomanjkljivosti ipd.), lahko postopek večkrat ponovi (Zanka 1 na diagramu). Ob potrditvi fotografije modul kamere posreduje fotografijo aplikacijski logiki. Ta naprej posreduje referenco, v pomnilniku shranjene fotografije modulu za obrezovanje fotografij (knjižnica Android Image Cropper). Ta prikaže nov grafični vmesnik, na katerem lahko uporabnik nastavi mere obreza fotografije. Po izbiri mer, torej po potrditvi obreza fotografije, dobi aplikacijska logika obrezano fotografijo. Nato se na zaslonu prikaže grafični vmesnik za OCR postopek. Ta je namenjen opozoritvi uporabnika o morebitnem dolgotrajnem postopku optičnega prepoznavanja znakov ter prikazu trenutnega stanja prepoznave (Zanka 2). Hkrati tudi požene Tesseract pogon nad obrezano fotografijo. Rezultat Tesseract-a je razpoznano besedilo v posebni XML obliki. Za tem se opravi postopek razčlenjevanja XML besedila. Sledi pregled obstoja razpoznanih besed v podatkovni bazi aditivov. Nato se uporabniku prikaže rezultat. Če obstajajo neprepoznane be-

sede, jih lahko uporabnik ročno popravi, kar sproži ponovno prepoznavanje aditivov ter prikaz novega rezultata (Zanka 3). Ta korak ni obvezen.

Velja omeniti, da postopek popravljanja besedila ni prisoten v optimalnem delovanju aplikacije. Optimalno delovanje pomeni najprej pravilno razpoznavo vseh sestavin iz fotografije in za tem še pomensko prepoznavo vseh sestavin v podatkovni bazi aditivov. Neoptimalno delovanje tako pomeni nepravilno razpoznavo vsaj ene sestavine iz fotografije ali neuspešno pomensko prepoznavo vsaj ene sestavine. Kot neoptimalno delovanje aplikacije seveda šteje tudi hkratna pojavitev obeh zgornjih primerov neoptimalnega delovanja. Neoptimalno delovanje aplikacije s seboj prinese kopico težav, zato je potreba po možnosti popravljanja besedila več kot očitna. Možni dogodki so naslednji:

- **Nepravilna razpoznava vsaj ene sestavine**

  Če pride do nepravilne razpoznave vsaj ene sestavine, obstajata dva možna primera težav.

  Prva težava se zgodi, kadar je rezultat razpoznave neobstoječa beseda oz. beseda, ki ne predstavlja kakšnega drugega aditiva (npr. besedo *lecitin* razpozna kot "podobno" besedo *lečitin* ali niz *l3citin* ali čisto drugačno besedo – *žoga* itn.).

  Druga težava je dogodek, ko je rezultat razpoznave beseda z dejanskim pomenom ter ta pomen predstavlja nek drug aditiv (npr. besedo *lecitin* razpozna kot besedo *lutein* ali besedo *tartrazin*, itn.).

  Razvidno je, da drugi primer lahko povzroči precej več nevšečnosti kot prvi, saj napačno razpoznana beseda predstavlja nek drug aditiv, ki ima lahko popolnoma drugačne lastnosti od dejanskega. Prav tako je odprava napak drugega primera precej zahtevnejša od prvega. Izkaže pa se, da je verjetnost za pojavitev te težave tako zelo zanemarljivo nizka, da se odprava le-te ne izplača. Če se bo le pojavila, jo bo pozoren uporabnik lahko opazil med pregledovanjem rezultata. Prvo težavo v tem koraku prav tako ignoriramo, saj bo odpravljena v naslednjem primeru.

- **Neuspešna pomenska prepoznava vsaj ene sestavine**

  Ponovno se srečamo z dvema možnostma:

  Prva možnost se nanaša na prvo težavo prejšnjega koraka, torej nepravilno razpoznana (nesmiselna) beseda. Te besede seveda ne bo v podatkovni bazi aditivov ter posledično ne bo pomensko prepoznana.

  Druga možnost je dogodek, ko je bila beseda pravilno razpoznana, torej govorimo o aditivu, ampak dotični aditiv ni bil najden v podatkovni bazi. To namiguje

na nepopolno podatkovno bazo in pomeni bodisi manjkajoči aditiv v podatkovni bazi ali manjkajoče sekundarno, terciarno ipd. ime aditiva v podatkovni bazi.

Ne poznam dovolj preprostega načina, kako programsko razlikovati med prvo in drugo možnostjo. Zato bo za obe možnosti veljala enaka rešitev – korak ročnega popravljanja besedila. Ta rešitev v celoti odpravi prvi problem ter zaobide drugi problem. Težava te rešitve je le v tem, da se za odpravo napak nanašam na uporabnika.

- **Obe hkrati**

  Seveda obstaja tudi možnost pojavitve obeh nevšečnosti hkrati, kjer bo daleč najbolj pogosta kombinacija obeh prvih možnosti, za katero je predvidena rešitev ročno popravljanje besed s strani uporabnika. Ostalim kombinacijam se bomo tako ali drugače izognili z vednostjo uporabnika. Vredno je ponovno opomniti, da je verjetnost za nastanek teh kombinacij (z izjemo prve) zelo nizka.

5.3 Načrtovanje posamičnih komponent

Sledijo načrti zahtevnejših komponent aplikacije. Mednje sodijo zajemanje ter obrezovanje fotografij, razpoznavanje besedila iz zajete fotografije, razčlenjevanje besedila, podatkovna baza, prepoznavanje zajetih besed, oblikovanje podatkov za prikaz ter funkcionalnost popravljanja besed.

5.3.1 Zajemanje fotografij

Zajemanje fotografij bom implementiral z uporabo privzete aplikacije za zajem fotografij (v nadaljevanju kamera) na uporabnikovi mobilni napravi. Android operacijski sistem namreč omogoča uporabo ostalih aplikacij naprave (če so te pravilno nastavljene) znotraj naše aplikacije za nekatere specifične namene. Ti nameni vključujejo na primer uporabo kamere, pošiljanje elektronske pošte (poštni odjemalec), predvajanje zvoka itd. Za uporabo uporabnikove kamere bom med klicanjem nove Android aktivnosti (angl. activity) uporabil naslednji namen (angl. intent) *ACTION_IMAGE_CAPTURE* iz razreda MediaStore. Slednje bo zagnalo kamero ter uporabniku omogočilo zajem fotografij. V svoji aplikaciji bom fotografijo prejel s pomočjo metode *onActivityResult(int requestCode, int resultCode, Intent data)*, kjer *data* predstavlja fotografijo, celoštevilski spremenljivki pa identifikator zahteve ter stanje rezultata (uspeh, napaka).

5.3.2 Obrezovanje fotografij

Za višjo uspešnost in kvaliteto prihodnjih faz je zajeto fotografijo običajno smiselno obrezati. Obrezovanje fotografij bo implementirano s pomočjo knjižnice *Android Image Cropper* različice 2.7.0. Knjižnico je potrebno najprej umestiti v projekt nato njene želene komponente inicializirati ter uporabiti. Poleg inicializacije grafičnega vmesnika bo potrebno implementirati še metodo za nastavitev prej zajete fotografije za obrez ter shranjevanje fotografije v želeni obliki.

5.3.3 Razpoznavanje besedila iz fotografije

ledi razpoznavanje besedila iz zajete ter obrezane fotografije. Proces razpoznavanja bo realiziran s knjižnico tess-two različice 9.0. Knjižnica tess-two je zbirka APIjev[2] ter orodij za uporabo Tesseract-a na Android sistemu ter predstavlja vmesnik za uporabo pogona Tesseract. Knjižnico je potrebno inicializirati (določitev jezikovne datoteke, določitev fotografije ter raznih parametrov). Po končani inicializaciji lahko kličemo metodo *String getUTF8Text()*za pridobitev razpoznanega besedila v navadni, tekstovni obliki. Klic omenjene metode seveda sproži proces razpoznavanja, zato podatki niso takoj na voljo. Android operacijski sistem je še posebej občutljiv na dolgotrajne oz. intenzivne operacije na glavni niti (angl. thread), zato bo potrebno proces razpoznavanja besed implementirati na svoji niti. Za to ne bom uporabil običajne nove Java niti, ampak Androidov t. i. *AsyncTask*. AsyncTask je namenjen obdelavi podatkov v ozadju ter objavi rezultata na glavni niti. In to je kot nalašč za moje potrebe.

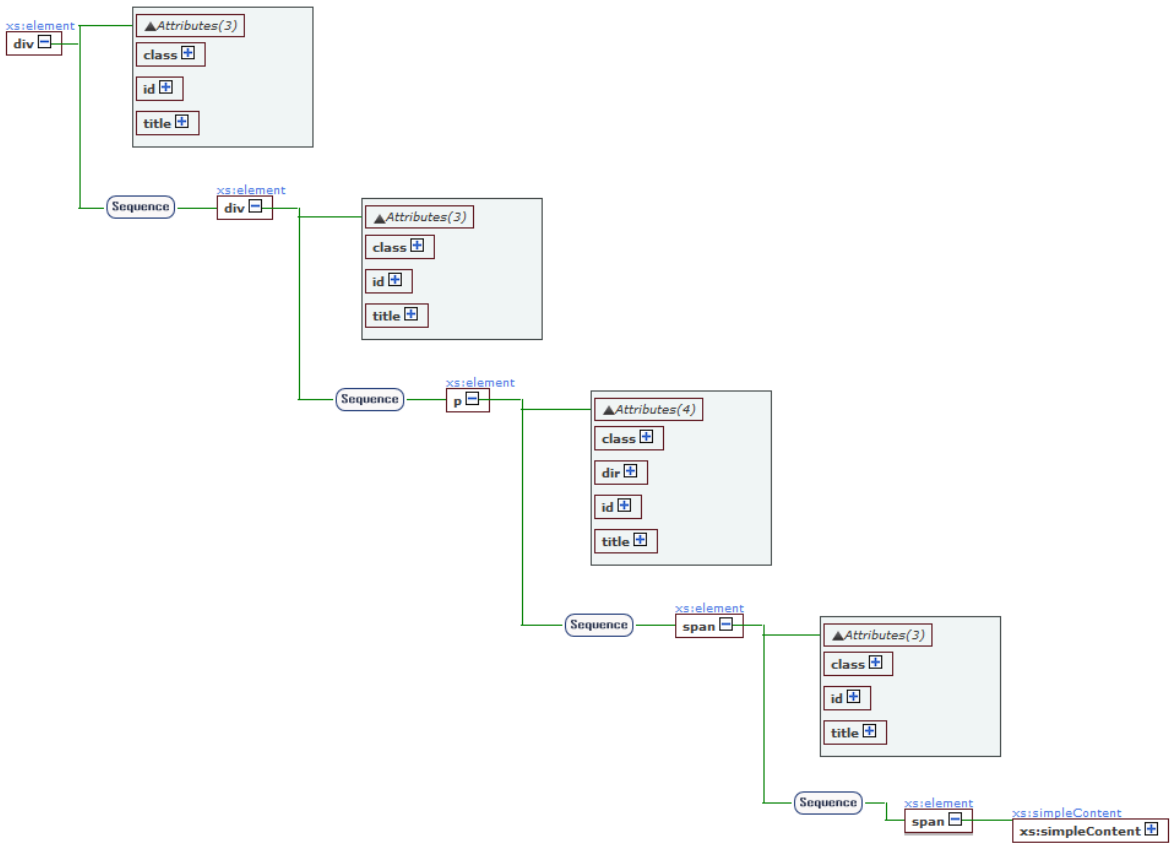
Prisotna je še metoda *String GetHOCRText(int pageNumber)*, ki je namenjena pridobitvi vseh podrobnosti o razpoznanem besedilu. Te podrobnosti so shranjene v hOCR formatu (XML format, prilagojen shranjevanju formatiranega besedila, prepoznanega z OCR tehnologijo). Podrobnosti vključujejo besedilo, stil, postavitev ter druge informacije [9]. Za potrebe razpoznavanja besed bom uporabil prav to metodo, saj vključuje podatke, ki jih bom potreboval pri nadaljnjem procesiranju.

5.3.4 Razčlenjevanje besedila

Kot že prej omenjeno, je hOCR besedilo shranjeno v obliki XML oz. XHTML ter ga je potrebno razčleniti v bolj uporabno obliko. Iz hOCR besedila bom pridobil podatke o postavitvi posameznih besed za potrebe označevanja aditivov v fazi prikaza rezultata in tudi podatek o zanesljivosti razpoznave besede. Primer podatkov je prikazan spodaj, delna hOCR shema pa na sliki 11.

[2]API - programski vmesnik (angl. application program interface)

```
<div class='ocr_page' id='page_1' title='image ""; bbox 0 0 2407 611; ppageno 0'>
  <div class='ocr_carea' id='block_1_1' title="bbox 61 0 2325 611">
    <p class='ocr_par' dir='ltr' id='par_1_1' title="bbox 61 0 2325 611">
      <span class='ocr_line' id='line_1_1' title="bbox 68 36 2321 167; baseline
          0.006 -37">
        <span class='ocrx_word' id='word_1_1' title='bbox 669 47 1009 136; x_wconf
            88' lang='slv' dir='ltr'>
          Sestavine:
        </span>
        <span class='ocrx_word' id='word_1_2' title='bbox 1036 50 1268 152; x_wconf
            83' lang='slv' dir='ltr'>
        sladkor,
        </span>
        <span class='ocrx_word' id='word_1_3' title='bbox 1301 52 1526 141; x_wconf
            86' lang='slv' dir='ltr'>
        koruzni
        </span>
        <span class='ocrx_word' id='word_1_4' title='bbox 1555 53 1737 156; x_wconf
            83' lang='slv' dir='ltr'>
        skrob,
        </span>
        <span class='ocrx_word' id='word_1_5' title='bbox 1768 56 2081 157; x_wconf
            83' lang='slv' dir='ltr'>
        dekstroza,
        </span>
        <span class='ocrx_word' id='word_1_6' title='bbox 2112 60 2321 167; x_wconf
            82' lang='slv' dir='ltr'>
        jedilna
        </span>
      </span>
    </p>
  </div>
</div>
```



Slika 11: Delna XSD shema hOCR formata.

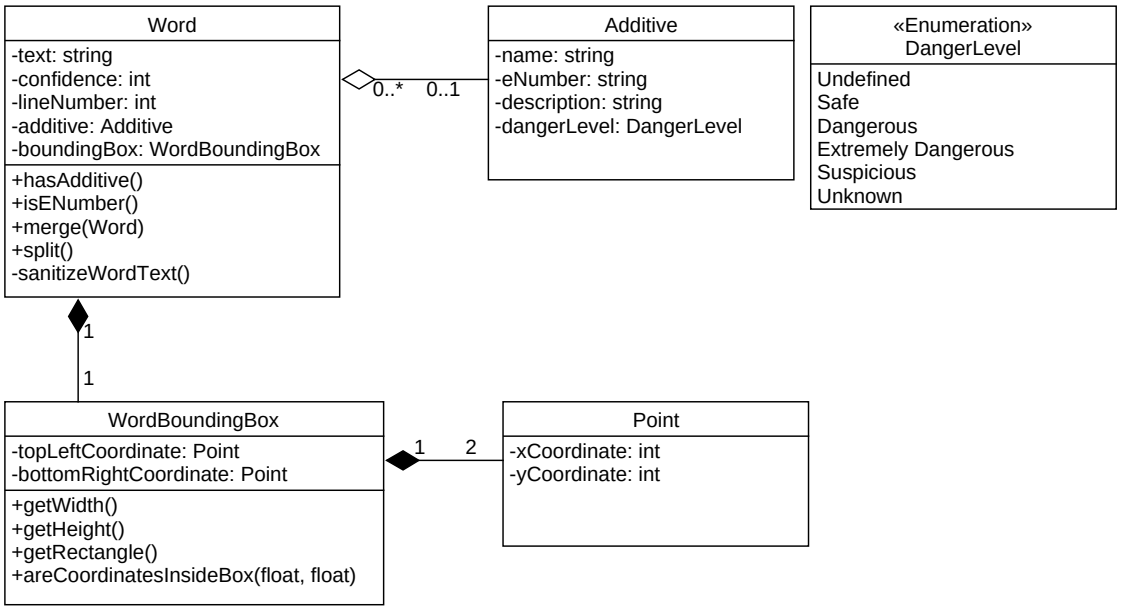
Načeloma me zanimajo le <*span*> značke, saj so v njih razpoznane besede ter njihove koordinate. Na primeru zgoraj opazimo dva tipa <*span*> značk. Ena ima atribut class (sl. razred) enak *ocr_line*, vse ostale pa imajo atribut class enak *ocrx_word*. Vsebina prve značke so vse besede v razpoznani vrstici besedila. Značka torej označuje razpoznano vrstico. Medtem je vsebina drugega tipa značk posamezna razpoznana beseda. Poleg atributa razred so tu še atributi id (sl. identifikator; hrani pa zapored vrstice oz. besede v besedilu), title (sl. naslov; hrani mejni okvir besede in stopnjo zaupanja pravilne razpoznave), lang (sl. jezik; hrani jezik razpoznanih besed) in dir (sl. smer; hrani usmerjenost besede - ltr pomeni left to right, torej iz leve proti desni). Druge značke so uporabne pri razčlenjevanju daljših, strukturiranih, besedil. Opazimo lahko prvo <*div*> značko z razredom *ocr_page*, ki označuje stran besedila. Druga <*div*> značka ima razred enak *ocr_carea* in označuje področje vsebine. Značka <*p*>, z razredom *ocr_par* označuje odstavek.

Za razčlenitev hOCR podatkov bom uporabil *XMLPull*[3] razčlenjevalnik. Z njim bom izluščil razpoznane besede in njihove lastnosti. Vsako besedo bom shranil v pripadajočo podatkovno strukturo, imenovano *Word*. Ta bo hranila besedo ter njene podrobnosti, kot so koordinate, zanesljivost optičnega razpoznavanja ter številko vrstice, na kateri beseda je. Omenjena struktura bo hranila tudi referenco na morebiten pripadajoči aditiv ter podobne informacije, ki se bodo izpolnile po prepoznavanju besede. Predvidena struktura tega dela sistema je prikazana na razrednem diagramu na sliki 12.

Na diagramu je razvidno, da ima vsaka beseda (Word) tudi nekaj metod, te so:

- hasAdditive() – preveri, ali je beseda aditiv oz. del aditiva (ali obstaja referenca na aditiv)
- isENumber() – preveri, ali je beseda E število. Implementacija je predvidena s pomočjo naslednjega regularnega izraza: ^(e|E)[1-9]\d{2,3}[(a-eA-E)]?$. Ta najprej preveri, ali se beseda začne z malo ali veliko črko e, nato preveri, ali ji sledi ena neničelna števka in za tem 2 do 3 poljubne števke. Za tem še preveri, ali se beseda zaključi s sicer neobvezno črko v razponu od a do vključno e (brez šumnikov).
- merge(Word) – združi podano besedo v trenutno; posledično se trenutnemu tekstu pripoji tekst druge besede; koordinate se ustrezno povečajo itn.
- split() – razdeli trenutno besedo glede na vpisane presledke v tekstu besede. Sorazmerno se tudi razdelijo koordinate.

[3]http://xmlpull.org



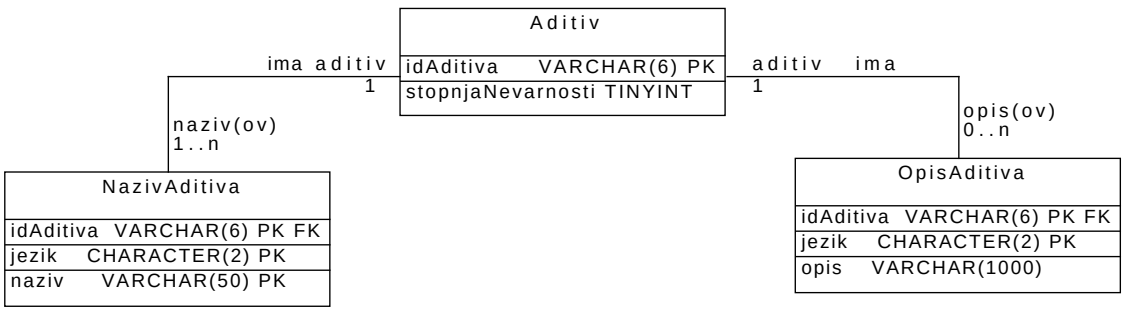
Slika 12: Razredni diagram podatkovne strukture za hranjenje podatkov o besedi.

- sanitizeWordText() – odstrani morebitne nečrkovne ter neštevilčne sekvence iz razpoznane besede.

5.3.5 Podatkovna baza

Za realizacijo podatkovne baze bom uporabil sistem za upravljanje s podatkovnimi bazami SQLite različice 3.14.1.

Predvideno strukturo podatkovne baze predstavlja spodaj prikazani relacijski diagram (slika 13). Za lažjo predstavo sem pripravil tudi primer zapisa podatkov poljubno izbranega aditiva. Ta je prikazan v tabelah 6, 8 ter 9.



Slika 13: Relacijski diagram predvidene podatkovne baze.

Tabela 6: Primer zapisa podatkov tabele Aditiv.

idAditiva *VARCHAR(6) PK*	**stopnjaNevarnosti** *TINYINT*
E101	0

Tabela *Aditiv*, prikazana v tabeli 6, služi hranjenju identifikatorjev (idAditiva) ter kategorij vseh aditivov. Za identifikator aditivov sem izbral t. i. *E število*, ki hkrati enolično določa aditiv ter njegov namen uporabe. Identifikator bom zapisoval s šestznakovnim (VARCHAR(6)) zapisom, kar omogoča shranjevanje zapisov dolžine 6 znakov v *UTF-8* kodiranju. E števila trenutno zajemajo vrednosti med E100 ter E1599. Včasih ima E število pripeto še črko. To je tudi razlog, zakaj sem se odločil za znakovni tip podatka namesto celoštevilskega. Identifikator bo seveda enoličen, torej ne bo dovoljeval podvajanja podatkov in bo predstavljal primarni ključ (angl. primary key, krajše PK) v tabeli.

Atribut stopnjaNevarnosti bo hranil stopnjo nevarnosti, ki jo aditiv predstavlja v primeru zaužitja. Hranjen bo v eno-bajtnem celoštevilskem (TINYINT) zapisu, ki omogoča vrednosti med 0 in 255. Besedne vrednosti stopenj nevarnosti ter z njimi povezanih opozoril za uporabnike bodo hranjene v mobilni aplikaciji, in sicer med lokalizacijskimi viri. Uporabljene vrednosti ter njihovi pomeni bodo naslednji:

Tabela 7: Stopnje nevarnosti ter njihov pomen.

Vrednost	**Pomen**
0	Aditiv ni nevaren
1	Škodljiv aditiv
2	Izredno škodljiv aditiv
3	Sumljiv aditiv
4	Neznano

Vrednost 0 bo torej predstavljala zdravju povsem neškodljive aditive. Vrednost 1 zdravju škodljive, a še vedno ne tako zelo nevarne aditive. Vrednost 2 bo predstavljala zdravju izredno nevarne oz. škodljive aditive, katerim se je priporočljivo popolnoma izogibati. Vrednost 3 bo predstavljala aditive, za katere obstaja sum o njihovi škodljivosti na zdravje. Medtem bo vrednost 4 predstavljala aditive, za katere ni znano oz. njihova (ne)škodljivost še ni dokazana.

Tabela *NazivAditiva*, prikazana v tabeli 8, je namenjena hrambi nazivov aditivov. Ti se bodo uporabljali predvsem med fazo prepoznavanja aditivov. V tabeli se ponovno

Tabela 8: Primer zapisa podatkov tabele NazivAditiva.

idAditiva *VARCHAR(6) PK FK*	**jezik** *CHARACTER(2) PK*	**naziv** *VARCHAR(50) PK*
E101	sl	riboflavin
E101	sl	laktoflavin
E101	sl	vitamin B2
E101	en	riboflavin
E101	en	vitamin B2

srečamo z identifikatorjem aditivov – idAditiva. Tokrat je njegov namen povezovanja s prejšnjo, glavno tabelo Aditiv. Povezava med tabelama je tipa 1 proti N s strani glavne tabele proti tabeli nazivov. To pomeni, da ima en aditiv iz glavne tabele lahko več pripadajočih nazivnih zapisov v nazivni tabeli. Ali drugače povedano, vsak aditiv ima lahko eno ali več imen. Podvajanje imen bi bilo seveda nesmiselno, zato vsi trije stolpci skupaj tvorijo primarni ključ.

Poleg identifikatorja sta v tabeli prisotna še atributa jezik ter naziv. Atribut jezik določa jezik, v katerem je podani naziv napisan. V zgornjem primeru se tako srečamo s slovenščino (sl) ter angleščino (en). Za hranjene jezika bom uporabil dvoznakovno *ISO 639-1*[4] kodo za zapis jezikov. Aplikacija oz. bolje rečeno prepoznavanje aditivov bo sicer trenutno na voljo le v slovenskem jeziku. Namen tega dodatnega atributa je morebitna razširitev na tuje trge v prihodnosti. Preostane le še atribut naziv, kateremu sem dodelil dolžino 50 znakov ter je namenjen hrambi nazivov aditivov.

Tabela 9: Primer zapisa podatkov tabele OpisAditiva.

idAditiva *VARCHAR(6) PK FK*	**jezik** *CHARACTER(2) PK*	**opis** *VARCHAR(1000)*
E101	sl	opis
E101	en	description

Tabela *OpisAditiva*, prikazana v tabeli 9, je namenjena hrambi opisov aditivov. Opisi so namenjeni uporabnikom v fazi prikaza rezultata, če želijo izvedeti več o posameznem aditivu. Tabela opisov je zelo podobna tabeli nazivov. Razlikuje se le v tem, da namesto naziva hrani opis. Za razliko od naziva je opis navadno daljši. Zato sem za posamezen opis aditiva določil maksimalno dolžino 1000 znakov. Povezava s tabelo Aditiv je zelo podobna povezavi med prejšnjima tabelama. Razlikuje se le v tem, da za vsak aditiv ne bo nujno obstajal tudi opis. Če pa bo, bo maksimalno en za vsak jezik.

[4]ISO standard za zapis jezikov

Ta lastnost je zagotovljena s tem, da idAditiva ter jezik skupaj predstavljata primarni ključ tabele.

Razlog za posebno obravnavo imen in opisov tiči v želji za preprečitev podvajanja večjih podatkov. Večji podatek je tu seveda opis (do 1000 znakov na zapis). Če bi vsak nazivni zapis vključeval hkrati tudi opis, bi se za vsak aditiv z vsaj dvema imenoma potrebna velikost vsaj podvojila. Tako pa podvajamo identifikator (6 znakov) ter jezik (2 znaka), kar je nedvomno boljša izbira.

5.3.6 Prepoznavanje zajetih besed

Korak razčlenjevanja besedila iz podpoglavja 5.3.4 pripravi seznam razpoznanih besed in njim pripadajočih lastnosti. Za vsako besedo bom preveril, ali je ta morda aditiv oz. ali obstaja možnost, da je ta del aditiva. Če je razpoznana beseda aditiv (v PB aditivov obstaja aditiv s prav takih nazivom oz. E številom, če je beseda E število), bom dotični besedi dodal referenco na njej pripadajoči aditiv ter se pomaknil na naslednjo besedo v seznamu.

Če pa se izkaže, da je beseda morebiti del aditiva (govora je o večbesednem aditivu), bom predpostavil, da je ta res aditiv ter pregledal ujemanje okoliških besed. Če se okoliške besede ne ujemajo s tistimi v predpostavljenem aditivu, razpoznana beseda najbrž ni aditiv oz. del aditiva ter nadaljujem z naslednjo besedo. Če se pa okoliške besede izkažejo za enake tistim v nazivu aditiva, bom vsaki besedi dodal referenco na dotični aditiv ter nadaljeval z naslednjo neprepoznano besedo.

Ker razpoznavanje besed ni dovolj zanesljivo, bom pri iskanju enakosti upošteval manjša odstopanja. Medbesedno odstopanje bom meril s t. i. *Levenshteinovo razdaljo*. Zelo preprosto rečeno je to število razlik med dvema besedama. Dejansko sprejemljivo vrednost odstopanja bom še določil med testiranjem. Lahko pa povem, da bo dinamično nastavljena glede na dolžino posamezne besede, saj bi fiksna vrednost povzročila več težav kot koristi.

Rezultat tega sklopa je dopolnjen začetni seznam besed. Besede, ki so aditivi oz. del aditivov, bodo hranile referenco na pripadajoči aditiv.

5.3.7 Oblikovanje podatkov za prikaz

Modul kot vhod dobi zgornji seznam z v najboljšem primeru vsemi prepoznanimi besedami, ki so aditivi oz. deli aditivov. Potreben je le še opis aditiva. Pridobitev tega pa je trivialna, saj vsaka prepoznana beseda hrani referenco na določen aditiv. Ta hrani tudi identifikator aditiva, tj. E število. Poizvedba bo izgledala nekako takole:

```
SELECT OpisAditiva.opis
FROM OpisAditiva
WHERE OpisAditiva.idAditiva =
'recognizedAdditivesList.get(i).eNumber'
AND Opis.jezik = 'LANG';
```

Po slovensko bi to pomenilo, IZBERI opis [aditiva] IZ tabele Aditiv OpisAditiva, KJER je identifikator aditiva tabele OpisAditiva enak E številu i-tega prepoznanega aditiva. Veljati pa mora tudi enakost med atributom jezik tabele Opis in jezikom nastavljenim v mobilni aplikaciji.

Po opravljenih poizvedbah nad vsemi prepoznanimi aditivi v seznamu imamo na voljo vse potrebne informacije za prikaz uporabniku – ime aditiva, E število, stopnjo nevarnosti, koordinate besede ter krajši opis.

Sledi označevanje na fotografiji, za kar bom uporabil *Canvas* ter *Paint* Java oz. Android razreda. Instanco Canvas razreda bom povezal s fotografijo tipa Bitmap. Nato bom nad instanco klical razne risalne metode, katere vhodi so koordinate ter lastnosti risanja v obliki Paint objekta. Tako bom vsaki stopnji nevarnosti priredil svoj Paint objekt, kateri se bodo med seboj razlikovali predvsem po barvi. Barve risanja bom izbiral glede na stopnjo nevarnosti posameznega aditiva. Približen rezultat označevanja aditivov je prikazan na sliki 4.

Preostane še prikaz dodatnih informacij, ki sem jih prej prejel iz podatkovne baze. Za vse te informacije ni prostora na zaslonu, zato se bodo prikazale s kliki na posamezne aditive na fotografiji. Za realizacijo tega potrebujem najprej rokovalnike dotikov na fotografiji. To so navadni rokovalniki dotika, ki so že implementirani v operacijskem sistemu Android. Dodal bom le prepoznavanje koordinat, tako da bom vedel, kateri aditiv je bil izbran (pritisnjen). Ob pritisku na poljuben aditiv se bo pojavil nov pogled z dodatnimi informacijami o izbranem aditivu (naziv, E število, tip, stopnja nevarnosti ter kratek opis). Primer tega pogleda je prikazan na sliki 5.

5.3.8 Popravljanje besed

V pogledu pregleda podrobnosti posamezne besede lahko uporabnik vsako besedo tudi popravi, če ugotovi, da je ta bila napačno razpoznana. Implementirani bodo trije načini

popravljanja besed. Predvidel sem tri možne tipe napak, in sicer:

- Beseda se razlikuje v nekaj nepraznih znakih (npr. k1slina namesto kislina).
- Dve ali več besed je bilo razpoznanih kot ena beseda (npr. citronskakislina namesto citronska kislina).
- Del besede je bil prepoznan kot samostojna beseda (npr. kisl ina namesto kislina).

Prvo bom rešil s preprosto možnostjo urejanja teksta (vnosno polje). Uporabnik spremeni napačno razpoznane znake ter potrdi spremembo.

Drugo bom rešil z možnostjo razcepitve besed. Če bo uporabnik v vnosno polje vnesel presledek med dvema nizoma, se bo zgodila razcepitev besede. Tu bo potrebno paziti na sorazmerno deljenje koordinat dotične besede.

Tretjo bom rešil z možnostjo združevanja besed. V pogledu podrobnosti besede bosta prisotna dva gumba za združevanje – levo ter desno. Ob pritisku na enega od njiju se bo trenutna beseda združila z levo ali z desno. Tu bo potrebno paziti na združevanje besed v različnih vrsticah.

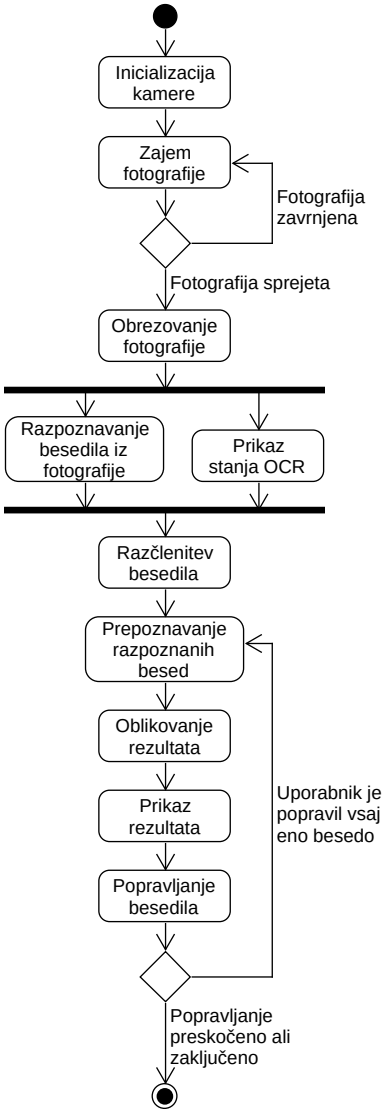
Ob vsaki potrjeni spremembi se bo znova izvedlo prepoznavanje zajetih besed ter primerno označilo morebitne nove prepoznane aditive oz. odstranilo oznake iz napačno zaznanih aditivov.

Optimalen rezultat tega koraka je pravilno razpoznan seznam vseh besed. Neoptimalen pa seznam, ki vsebuje napačno razpoznane besede, kar lahko vpliva na popolnost končnega rezultata – nepreverjeni aditivi. Morebitna težava je tudi v tem, da ima v tem koraku uporabnik precej proste roke, kar lahko privede do še manj zanesljivega rezultata. Na primer, besedo, ki definitivno ni aditiv, popravi v besedo, ki to je itn. Nenamerne napake lahko poizkusim omiliti s tem, da uporabniku priporočim morebiten popravek – kar se sicer do neke mere že samodejno dogaja v fazi prepoznave aditivov – lahko bi v tem koraku toleranco podobnosti besed nekoliko povečal. Za namerne napake pa trenutno ne poznam pametne rešitve. Zdi se mi pa samoumevno, da z namerno napačno uporabo ni smotrno pričakovati zanesljivega rezultata.

Predviden grafični vmesnik je predstavljen na sliki 5.

5.4 Potek izvajanja aplikacije

Potek izvajanja najbolje predstavimo z diagramom aktivnosti. Diagram nazorno prikaže posamezne programske korake, umeščene v celotno sekvenco izvedbe programa. Posledično prikazuje tudi kontrolni oz. delovni tok programa. Z diagramom aktivnosti lahko tudi pregledno prikažemo izbirne vejitve, vzporednost ter iteracije [10]. Diagram aktivnosti na sliki 14 prikazuje predvideno delovanje načrtovane aplikacije.



Slika 14: Diagram aktivnosti mobilne aplikacije.

Potek izvedbe aplikacije je naslednji:

1. Ob zagonu aplikacije uporabnik sproži modul kamere.
2. Uporabnik zajame in potrdi fotografijo.
   - 2.1. Alternativni potek: Uporabnik zajame in zavrne fotografijo ter ostane v koraku zajemanja fotografije.
3. Uporabnik obreže zajeto fotografijo.
4. Prične se razpoznavanje besedila iz obrezane fotografije; hkrati se tudi prikaže stanje ter redno osvežuje stanje OCR postopka.
5. Izvede se razčlenjevanje razpoznanega besedila.
6. Sproži se prepoznavanje razpoznanih besed.
7. Začne se korak oblikovanja rezultata.
8. Na zaslonu se prikaže rezultat.
9. Sledi korak popravljanja besedila (aditivov), ki je v optimalnih pogojih (vsi aditivi prepoznani) nepotreben.
   - 9.1. Alternativni potek: Uporabnik se odloči popravljati nepravilno razpoznane besede.
     - 9.1.1. Uporabnik popravi neprepoznano besedo.
     - 9.1.2. Aplikacija se vrne v stanje 6.
   - 9.2. Alternativni potek: Uporabnik preskoči popravljanje.

6 Izvedba

Načrtovanju sledi faza izvedbe. Ta je namenjena pripravi končnega programskega izdelka. Med njegovo pripravo moramo upoštevati rezultate prejšnjih faz, predvsem pa faze načrtovanja. Med izvedbo običajno tudi odkrijemo napake in pomanjkljivosti prejšnjih faz, če smo te storili. Te pa znajo biti usodne in lahko pomenijo tudi predčasno prekinitev projekta, torej neuspeh, in v bolj pozitivnih primerih pripravo nepopolnega izdelka. Nepopoln izdelek je potrebno popraviti, kar pa lahko privede do novih težav. Najbrž ni potrebno izgubljati besed o nastali škodi, predvsem finančni, ki utegne biti zelo visoka. Med izvedbo oz. po uspešno zaključeni izvedbi šele ugotovimo ogromen pomen vseh prejšnjih faz.

To je torej faza, kjer se bom lotil programiranja. V tem delu dokumenta bom predstavil pomembnejše tehnologije, ki jih bom tako ali drugače uporabil.

6.1 Uporabljene tehnologije

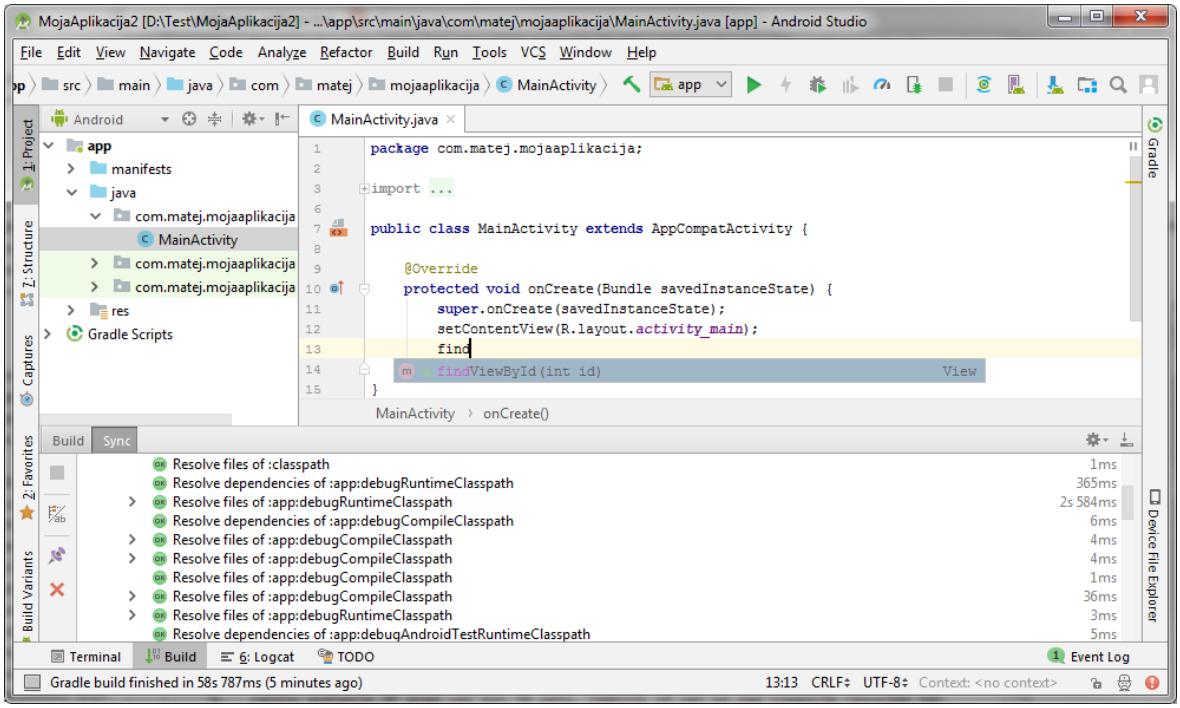
Poglavje je namenjeno krajši predstavitvi nekaterih tehnologij, s katerimi bom imel opravka. Mednje sodijo Android Studio razvojno okolje, Java programski jezik, Tesseract OCR knjižnica ter SQLite sistem za upravljanje podatkovnih baz.

6.1.1 Android Studio

Za razvoj aplikacije bom uporabil Android Studio, ki ga je razvilo podjetje Google. Android Studio je t. i. integrirano razvojno okolje (angl. integrated development environment ali IDE). Namenjeno je razvijanju Android aplikacij v programskih jezikih Java ter po novem tudi Kotlin. Tako kot ostala integrirana razvojna okolja nam tudi Android Studio zelo olajša razvoj aplikacij z razno avtomatizacijo. Poleg tega vključuje tudi preprost urejevalnik grafičnega vmesnika, podporo sistemom za upravljanje z izvorno kodo, enostavno upravljanje lokalizacije aplikacij ipd.

Za pravilno delovanje okolja sta potrebna še *Java Development Kit* (uporabljena različica 1.8) ter paket za razvoj programske opreme Android, imenovan *Android SDK*. Za sprotno preverjanje delovanja spisane kode je potreben še Android posnemovalnik (angl. emulator) ali dejanska Android naprava nastavljena na razvijalski način. Android posnemovalniki žal niso znani po svoji hitrosti in odzivnosti, zato bom za potrebe

razvoja raje uporabil svoj mobilni telefon. Gre za pametni mobilnik Sony Xperia XA2 z nameščenim operacijskim sistemom Android različice 8.0.



Slika 15: Zaslonski posnetek glavnega zaslona v Android Studio.

6.1.2 Java

Java je najverjetneje trenutno najbolj uporabljan objektni programski jezik. Je tudi primarni jezik Android platforme, kar je tudi razlog za mojo uporabo Jave. Za poganjanje Java aplikacij potrebujemo Java Virtual Machine (Java navidezni stroj), ki je navadno zapakiran v programski paket Java Runtime Environment (Java zagonsko okolje). Za razvoj Java aplikacij potrebujemo zgoraj omenjeni Java Development Kit, ki vsebuje poleg zagonskega okolja še prevajalnik programske kode, orodje za upravljanje podatkovnih baz, orodje za dokumentiranje itn.

6.1.3 Tesseract

Tesseract je odprtokodni pogon za optično prepoznavanje znakov. Razvoj nad pogonom je začelo podjetje HP pred več kot 30 leti. Zadnjih 10 let nad njegovim razvojem bdi podjetje Google. Napisan je v programskih jezikih C ter C++ ter podpira glavne namizne platforme (Windows, Linux, Mac OS) [5]. Ker sam po sebi ne podpira operacijskega sistema Android, sem moral izbrati enega od vmesnikov (tess-two) za njegovo uporabo na Android platformi.

6.1.4 SQLite

SQLite je odprtokodni sistem za upravljanje relacijske podatkovne baze. Posebnost SQLite je v tem, da ne deluje po principu odjemalec-strežnik, temveč se neposredno vgradi v končni program, kar pomeni direkten dostop do podatkovne baze [11]. Ta lastnost zelo poenostavi in pohitri upravljanje s podatkovno bazo. Poleg tega je namen sistema tudi kompaktnost, kar potrdi podatek, da je najmanjša možna velikost celotnega sistema le pičlih 500 KB [6]. To so razlogi za izbiro prav te rešitve.

6.2 Podrobnosti izvedbe

Sledijo predstavitve zanimivejših konceptov izvedbe.

6.2.1 Uporaba tess-two knižnice

Večino odprtokodnih knjižnic za Android platformo lahko zelo enostavno vključimo v projekt. To se stori tako, da se v datoteko *build.gradle* vnese referenco implementacije dotične knjižnice. Sledi primer vključitve tess-two knjižnice različice 9.

```
dependencies {
  implementation 'com.rmtheis:tess-two:9.0.0'
}
```

Po vnosu reference se projekt sinhronizira ter prenese dodane knjižnice. Te so sedaj pripravljene za uporabo.

Uporaba tess-two knjižnice je v osnovi precej preprosta. Kliče se jo na naslednji način:

```
TessBaseAPI ocr = new TessBaseAPI(progressNotifier);
ocr.init(datapath, language);
ocr.setImage(image);
String hocrResult = ocr.getHOCRText();
ocr.end();
```

Najprej se instancira *TessBaseAPI* vmesnik za uporabo Tesseract pogona. Temu se lahko poda poslušalec (angl. listener) napredka OCR procesa, ki je namenjen npr. prikazovanju napredka na zaslonu. Sledi inicializacija vmesnika – potrebno mu je podati pot do jezikovne datoteke ter določiti jezik prepoznave. Za tem se določi slika, nad katero se bo izvedlo razpoznavanje besedila. S klicem metode *ocr.getHOCRText()* se dejansko sproži proces razpoznave, ki je glede na vnos lahko dolgotrajen proces. Ko

proces zaključi, vrne razpoznano besedilo v hOCR obliki. Potrebno je le še sprostiti vire s klicem metode *ocr.end()*.

Kot sem omenil, je proces razpoznave lahko dolgotrajen, zato je potrebno zagotoviti asinhrono implementacijo. Tako sem zgornji izvleček kode ovil v Androidovo asinhrono opravilo (angl. asynchronous task). V osnovi imajo taki asinhroni klici tri korake, in sicer predobdelavo, obdelavo (v ozadju), ter poobdelavo. V predobdelavi preverim, ali jezikovna datoteka obstaja ter v primeru neobstoja, le-to kopiram iz aplikacijskega paketa v uporabniški prostor naprave. V obdelavi izvedem proces razpoznave besedila. V poobdelavi pa posredujem rezultat obdelave naslednjemu koraku (prepoznavanje aditivov).

6.2.2 Izračun Levenshteinove razdalje

Kot sem že omenil v poglavju 5.3.6 – prepoznavanja zajetih besed, proces razpoznavanja ni vedno zanesljiv. Za delno omilitev negativnih posledic nezanesljive razpoznave sem implementiral primerjalnik besed in aditivov, ki zanemarja manjša odstopanja. Za računanje odstopanj sem uporabil prav t. i. Levenshteinovo razdaljo. Implementirana je iterativno z matriko.

```
public static int calculate(String first, String second) {
  int[][] distance = new int[first.length() + 1][second.length() + 1];

  for (int i = 0; i <= first.length(); i++) {
    distance[i][0] = i;
  }
  for (int j = 1; j <= second.length(); j++) {
    distance[0][j] = j;
  }

  for (int i = 1; i <= first.length(); i++) {
    for (int j = 1; j <= second.length(); j++) {
      distance[i][j] = minimum(
        distance[i - 1][j] + 1,
        distance[i][j - 1] + 1,
        distance[i - 1][j - 1]
        + ((first.charAt(i - 1) == second.charAt(j - 1))? 0 : 1)
      );
    }
  }

  return distance[first.length()][second.length()];
}
```

Metoda *calculate* sprejme dve besedi ter izračuna razdaljo med njima. Postopek se prične s pripravo celoštevilskega dvodimenzionalnega polja (matrike) velikosti dolžine prve besede povečane za enkratno dolžino druge besede povečane za 1.

Sledi polnjenje zgornjega ter levega roba matrike z indeksi enakimi poziciji, na kateri so, npr. polje z indeksoma (1, 0) bo dobilo vrednost 1, medtem ko bo polje z

indeksoma (0, 4) dobilo vrednost 4. Lahko si predstavljamo, da vsaka od omenjenih stranic predstavlja eno od besed. Trenutno vpisane vrednosti predstavljajo število potrebnih operacij za pripravo ene ali druge besede iz ničesar. Ta zamisel je ponazorjena na spodnji tabeli (tabela 10) s poljubnima besedama list ter čist kot primer.

Tabela 10: Prvotni izgled matrike za besedi list ter čist.

		l	**i**	**s**	**t**
	0	1	2	3	4
č	1				
i	2				
s	3				
t	4				

Za tem se začne izvajati glavni del algoritma, to sta dve *for* zanki, katerih proizvedeni indeksi ustrezajo še ne izpolnjenim poljem matrike. Do teh indeksov dostopamo v notranji for zanki, kjer se trenutnima indeksoma tudi računa pripadajoča vrednost. Izračun poteka po naslednjem algoritmu, kjer *distance[i - 1][j] + 1* predstavlja izbris znaka, *distance[i][j - 1] + 1* dodajanje znaka in *((first.charAt(i - 1) == second.charAt(j - 1))? 0 : 1)* (ne)ujemanje znaka prve (*first*) ter druge (*second*) besede. Vzame se najmanjšo od treh vrednosti, saj računamo najkrajšo razdaljo med besedama.

```
distance[i][j] = minimum(
    distance[i - 1][j] + 1,
    distance[i][j - 1] + 1,
    distance[i - 1][j - 1]
    + ((first.charAt(i - 1) == second.charAt(j - 1))? 0 : 1)
);
```

Po zaključenem izračunu je matrika napolnjena (primer na tabeli 11; iz nje lahko razberemo najkrajšo razdaljo med besedama. Ta je v matriki na paru indeksov, ki ustrezata dolžini obeh besed. Za predstavljen primer bi to bil par indeksov 4 (dolžina besede list), 4 (dolžina besede čist). Vrednost pa ustreza številu 1, kar pomeni, da je potrebna le ena operacija za preoblikovanje ene besede v drugo, kar je res, saj se besedi razlikujeta le v prvi črki.

Tabela 11: Končni izgled matrike za besedi list ter čist.

		l	**i**	**s**	**t**
	0	1	2	3	4
č	1	1	2	3	3
i	2	2	1	2	3
s	3	3	2	1	2
t	4	4	3	2	1

7 Testiranje

Izvedbi sledi testiranje. Odločil sem se testirati po t. i. "box" pristopu, ki proces razdeli na dva dela - white box ter black box (sl. strukturni ter vedenjski tip testiranja). Opravil sem tudi test nefunkcijskih zahtev na vzorcu testnih uporabnikov. Opisi testov ter rezultati sledijo v naslednjih poglavjih.

Vredno je tudi omeniti, da je testiranje, vsaj v formalni obliki razvoja, skrbno načrtovan ter dolgotrajen proces. Potrebni so podrobni opisi oz. načrti vsakega testnega primera – začetno stanje, postopek testiranja, morebitni vhodni podatki, testno okolje in navsezadnje tudi pričakovan rezultat.

V sklopu te zaključne naloge sem fazo testiranja zelo omejil zaradi časovne stiske. Posledično sem le delno opravil testiranje (predvsem za ugotavljanje osnovnega delovanja aplikacije).

7.1 White box (strukturno testiranje)

White box oz. strukturno testiranje predstavlja okolje, v katerega imamo vpogled. To pomeni, da poznamo npr. izvorno kodo ter strukturo programa, njegov podatkovni ter kontrolni tok ipd [12]. V tem stadiju torej, preprosto povedano, testiramo delovanje kode programa. Zato je smiselno, da dotične teste pripravi razvijalec, ki dobro pozna kodo.

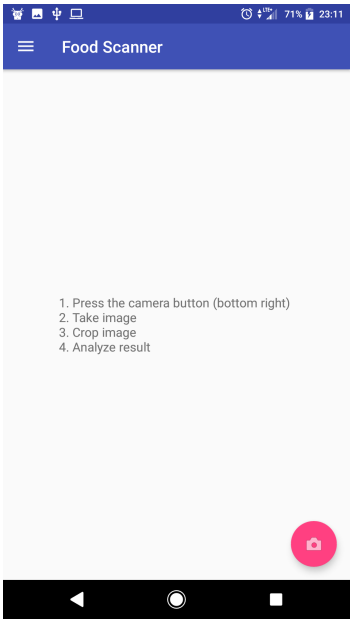
Najpogosteje to testiramo s pomočjo enotskih (angl. unit), integracijskih ter regresijskih testov. Testiranje enot je namenjeno testiranju posameznih enot ali skupin povezanih enot. Integracijski testi so namenjeni testiranju interakcije med posameznimi komponentami. Regresijski testi so namenjeni testiranju spremenjenih komponent, če spremembe niso prinesle neželenih posledic [13].

Zaradi pomanjkanja časa white box testiranja nisem opravil v celoti. Pripravil sem le nekaj unit testov, ki testirajo delovanje nekaterih kritičnih metod. Teste sem napisal s pomočjo JUnit ogrodja, ki je praktično standardno ogrodje za pripravo unit testov v programskem jeziku Java.

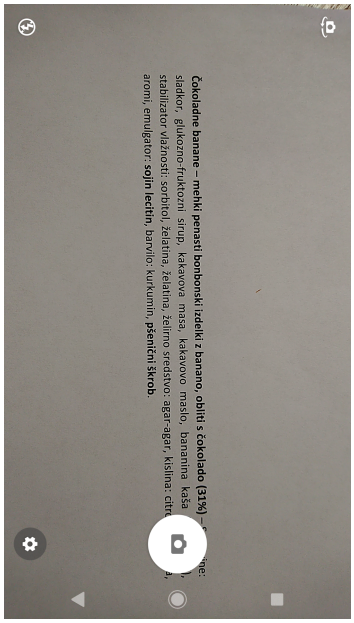
7.2 Black box (vedenjsko testiranje)

Medtem ko white box testiranje predstavlja poznano okolje, black box oz. vedenjsko testiranje predstavlja ravno nasprotno – okolje katerega notranjega delovanja sploh ne poznamo, ter nas ta konec koncev tudi ne zanima. Kar lahko v taki situaciji testiramo, je torej funkcionalnost programa. Za opravljanje takšnega tipa testiranja ni potrebno poznavanje kode ali pa znanje razvijanja [14]. Za kvalitetno testiranje je seveda potrebna velika mera natančnosti ter sistematičnosti. Tokrat sem sicer opravil dotično testiranje sam.

7.2.1 Zajemanje fotografije



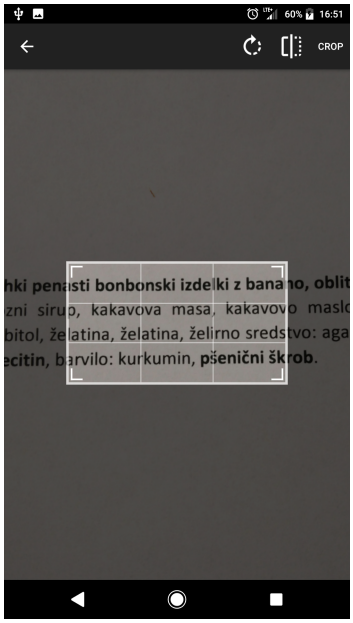
Slika 16: Začetni zaslon.



Slika 17: Modul kamere.

Na glavnem pogledu (slika 16) sem pritisnil na gumb za zagon modula kamere. Kamera se je prikazala; uspešno sem zajel želeno fotografijo.

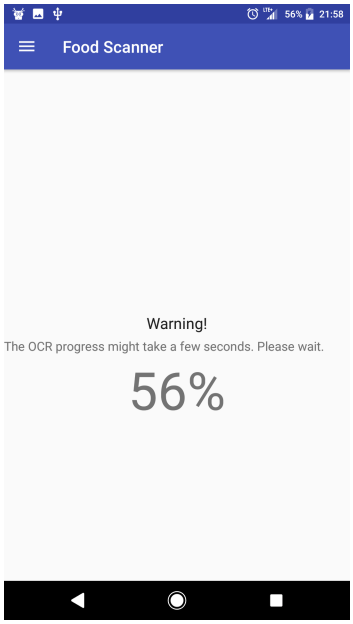
7.2.2 Obrezovanje zajete fotografije



Slika 18: Modul obrezovanja zajete fotografije.

Po uspešnem zajetju fotografije se samodejno zažene modul obrezovanja le-te. Fotografiji sem uspešno nastavil ter potrdil mere obreza.

7.2.3 OCR postopek

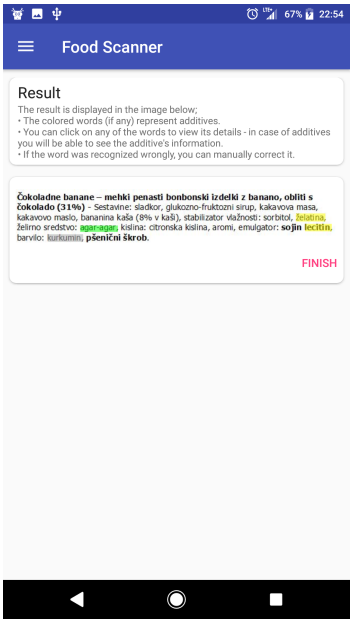


Slika 19: Pogled s podatki o trajanju OCR postopka.

Potrditev mer obreza zažene OCR postopek. Ob tem se prikaže zaslon, ki sproti izpisuje stanje OCR procesa. Tudi ta je deloval kot zastavljeno.

7.2.4 Pregled rezultata

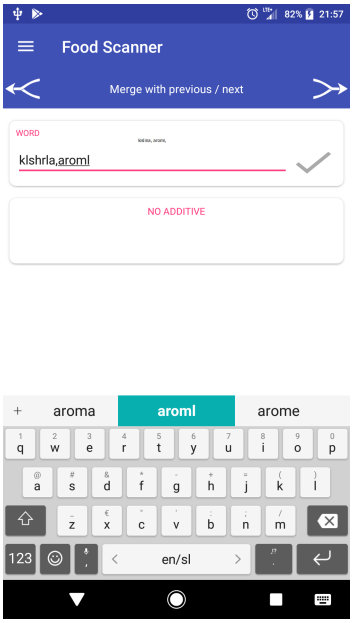
Po končanem OCR procesu se prikaže zaslon z rezultatom (slika 20). Tukaj me je pričakal rezultat s štirimi pravilno prepoznanimi aditivi. Nemudoma sem tudi opazil vsaj en aditiv, ki ni bil pravilno razpoznan – citronska kislina. Pritisnil sem tudi na enega od prepoznanih aditivov ter uspešno pregledal njegove podrobnosti (slika 21).



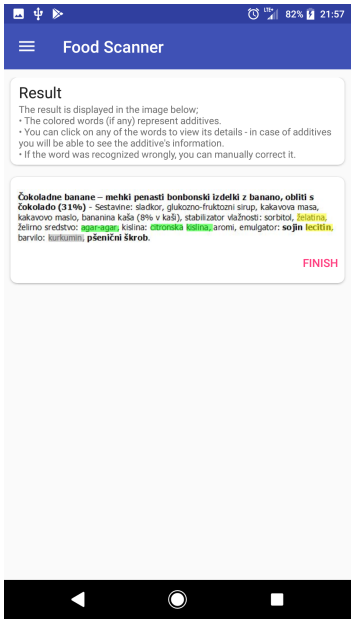
Slika 20: Pregled rezultata.

Slika 21: Pregled prepoznanega aditiva.

7.2.5 Popravljanje besed



Slika 22: Popravljanje napačno razpoznane besede.



Slika 23: Pregled popravljenega rezultata.

Napačno razpoznano besedo oz. aditiv sem ročno popravil (slika 22), kar je ponovno sprožilo tudi proces prepoznavanja aditivov. Rezultat je že takoj opazen, saj je tokrat aditiv bil uspešno prepoznan (slika 23).

7.3 Testiranje na vzorcu uporabnikov

Kot nefunkcijske zahteve mobilne aplikacije sem v poglavju 4.3 Nefunkcijske zahteve omenil enostavnost, preglednost, odzivnost ter dostopnost. Vse razen zadnje sem preveril s povratnim odzivom testne skupine. Testno skupino so predstavljale različno tehnično podkovane osebe z različnimi mobilnimi napravami.

7.3.1 Enostavnost

Testni uporabniki so bili enotnega mnenja, da je aplikacija zelo enostavna za uporabo. Nekateri so sicer omenili, da je obrezovanje fotografij nekoliko nerodno, s čimer se tudi jaz strinjam. Nevšečnost izvira iz knjižnice za obrezovanje fotografij; izboljšava le-te pa bi potrebovala veliko časa, zato sem trenutno odpravo le-te odložil do nadaljnjega.

7.3.2 Preglednost

Glede preglednosti je bilo nekaj več pripomb. Večina testnih uporabnikov je pogrešala sliko z besedo v pogledu podrobnosti besed za lažjo primerjavo med dejansko ter razpoznano besedo. Pripomba se mi zdi smiselna, zato sem se jo odločil odpraviti.

Nekateri so pa tudi pogrešali še možnost povečevanja (angl. zoom) označene fotografije rezultata, saj ta utegne biti pri večjih fotografijah zelo pomanjšana, kar otežuje pregled ter klikanje po posameznih besedah. Povečava rezultata se mi zdi zelo pomembna, saj je pregled le-tega glavni cilj uporabe aplikacije, zato bom tudi to funkcionalnost poizkusil implementirati.

7.3.3 Odzivnost

Večina testnih uporabnikov je bila zadovoljna z odzivnostjo aplikacije. Nekateri s šibkejšimi napravami so se sicer pritoževali glede trajanja postopka OCR, kjer pa možnosti optimizacije nimam prav veliko. Možna alternativa bi bilo opravljanje OCR procesa v oblaku, kar pa trenutno predstavlja prevelik zalogaj, saj bi potreboval še strežnik ter kar nekaj časa za razvoj.

Pojavil se je tudi predlog ročne predčasne ustavitve OCR procesa, če ta traja predolgo. Ta predlog se mi prav tako zdi na mestu, zato bom poizkusil implementirati tudi to funkcionalnost.

8 Zaključek

Zaključna naloga predstavlja formalni proces programskega inženirstva nad manjšim, praktičnim, primerom. Omenjena majhnost projekta ter časovne omejitve (predvsem bi si želel nekoliko več časa za pripravo aplikacije) so povzročile, da je taisti proces izšel v nekoliko okrnjeni različici. Pri večjih projektih bi vsekakor bila potrebna večja natančnost in obsežnost.

Med fazo izvedbe sem namreč dobil nemalo zamisli o možnih izboljšavah in razširitvah. Med drugim bi poizkusil skrajšati število korakov uporabe aplikacije. To bi storil z združitvijo koraka zajemanja fotografije ter obrezovanja fotografije. Moral bi pripraviti kamero po meri, ki bi na zaslonu prikazovala razširljiv pravokotnik. Fotografijo bi tako hkrati zajeli ter obrezali.

Razmišljal sem tudi o možnosti razpoznave besedila v oblaku – namenjeno predvsem za šibkejše naprave oz. za natančnejšo razpoznavo besedila. Ta ideja potrebuje namenski strežnik, na katerega bi aplikacija poslala obrezano fotografijo. Nad njo bi se na strežniku izvedel proces razpoznave besedila. Razpoznano besedilo bi se vrnilo aplikaciji. Na namenskem strežniku bi lahko tudi uporabil novejšo različico Tesseract pogona, ki prinaša obilico izboljšav v primerjavi s starejšim. Nekaj časa bi seveda tudi rad namenil izboljšanju izgleda aplikacije, saj je zdajšnji izgled precej osnoven.

Ne glede na zgoraj napisano sem s projektom zadovoljen. Vloženega je bilo namreč bilo veliko truda in časa. Prototipni izdelek je tudi že kolikor toliko uporaben. Pri pripravi projekta sem se tudi veliko novega naučil.

9 Literatura

[1] M. Peterman, T. Pajk Žontar in D. Pondelek, *Aditivi*. Zveza Potrošnikov Slovenije, Ljubljana, 2007. *(Citirano na strani 1.)*

[2] J. Gams, *O projektu*, `http://ninamvseeno.org/projekt.aspx`. (Datum ogleda: 9. 6. 2018.) *(Citirano na strani 2.)*

[3] A. Teplitzki, *Android Image Cropper*, `https://arthurhub.github.io/Android-Image-Cropper/`. (Datum ogleda: 10. 6. 2018.) *(Citirano na strani 4.)*

[4] *Tess-two*, `https://github.com/rmtheis/tess-two`. (Datum ogleda: 10. 6. 2018.) *(Citirano na strani 4.)*

[5] *Tesseract (Software)*, `https://en.wikipedia.org/wiki/Tesseract_(software)`. (Datum ogleda: 10. 6. 2018.) *(Citirano na straneh 4 in 39.)*

[6] *SQLite*, `https://www.sqlite.org`. (Datum ogleda: 11. 6. 2018.) *(Citirano na straneh 4 in 40.)*

[7] *View*, `https://developer.android.com/reference/android/view/View`. (Datum ogleda: 11. 6. 2018.) *(Citirano na strani 4.)*

[8] C. Larman, *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*, Addison Wesley, Third Edition, 2004. *(Citirano na strani 20.)*

[9] T. Breuel, *The hOCR Embedded OCR Workflow and Output Format*, 2010. *(Citirano na strani 27.)*

[10] P. Rogelj, *Skripta za predmet programsko inženirstvo*, Koper, 2014, Prva izdaja. *(Citirano na straneh 19 in 36.)*

[11] *SQLite*, `https://en.wikipedia.org/wiki/SQLite`. (Datum ogleda: 11. 7. 2018.) *(Citirano na strani 40.)*

[12] K. A. Saleh, *Software Engineering*, J. Ross Publishing, 2009. *(Citirano na strani 44.)*

[13] L. Williams, *White-Box Testing*, `https://students.cs.byu.edu/~cs340ta/fall2017/readings/WhiteBox.pdf`. (Datum ogleda: 25. 7. 2018.) *(Citirano na strani 44.)*

[14] M. G. Limaye, *Software Testing*, Tata McGraw-Hill Education, 2009. *(Citirano na strani 45.)*

Priloge

A Izvorna koda aplikacije

Zaključni nalogi je priložena zgoščenka z izvorno kodo mobilne aplikacije.