Nea Valtonen

Lääkepakkausmateriaalien käsittelysovellus

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

16.11.2018

Metropolia

Tekijä Otsikko Sivumäärä Aika	Nea Valtonen Lääkepakkausmateriaalien käsittelysovellus 35 sivua 16.11.2018
Tutkinto	Insinööri (AMK)
Tutkinto-ohjelma	Tieto- ja viestintätekniikka
Ammatillinen pääaine	Ohjelmistotuotanto
Ohjaajat	IT-asiantuntija Ari Oinonen Yliopettaja Erja Nikunen

Työn tavoitteena oli tehdä Lääketietokeskus Oy:lle sovellus, joka automatisoi lääkkeiden ulkopakkausmateriaalien käsittelyprosessia ja tekee prosessista sujuvaa ja tehokasta henkilöstölle. Tehtävän sovelluksen tarkoituksena on tunnistaa ja poimia lääkepakkauksien sivut olemassa olevista materiaaleista, jotka sisältävät muun muassa lääkepakkauksen levityskuvan.

Sovellus toteutettiin kolmeen eri Visual Studion projektiin käyttämällä C#-ohjelmointikieltä. Ensimmäinen VS-projekteista hakee rajapinnasta dataa ja tallentaa haetun datan tietokantaan. Toinen VS-projekti tehtiin kirjastoprojektiksi, jonne tehtiin ulkopakkausmateriaalien käsittelyyn käytettävät funktiot. Kolmas VS-projekti oli sovelluksen web-käyttöliittymä, joka kutsui kirjastoprojektin funktioita.

Insinöörityön lopputuloksena syntyi toimiva sovellus, joka täyttää sille annetut tavoitteet. Sovellus poimii materiaaleista lääkepakkauksen sivut ja esittää käyttöliittymässä käsittelyprosessin tulokset käyttäjälle. Käyttöliittymässä käyttäjä pystyy hallinnoimaan jo lisättyjä ulkopakkauksia sekä lisäämään uusia. Sovellusta aiotaan vielä jatkokehittää ennen sen tuotantoon siirtämistä.

Avainsanat	Tekstintunnistus, Kuvankäsittely, AForge, Tesseract

Author Title Number of Pages Date	Nea Valtonen Handling Application for Medicine Packaging Materials 35 pages 16 November 2018
Degree	Bachelor of Engineering
Degree Programme	Information and communication technologies
Professional Major	Software engineering
Instructors	Ari Oinonen, IT Specialist Erja Nikunen, Principal Lecturer

The aim of this thesis was to create an application for Lääketietokeskus Oy to automate the handling process of the outer packaging of medicines and make the process smooth and efficient for its users. The goal of the application is to identify and extract the sides of medicine package from existing materials, such as the spreading image of the medicine package.

The application was implemented in three different Visual Studio projects using C# -programming language. The first VS project retrieves data from the interface and stores the requested data into the database. The second VS project is a library project containing all of the functions used to handle outer packaging materials. The third VS project is application's web user interface, which uses library project's functions.

As a result of this thesis a working application was developed that fulfils all goals. The application extracts the sides of the medicine package from the source material and presents the processed results to the user in the user interface. In the user interface, user can manage the existing outer packages and add new ones. The application will be further developed before it is put into production.

Keywords	OCR, Image Processing, AForge, Tesseract

Sisällys

Lyhenteet

1 Johdanto 1

2 Tietokonenäkö 1

2.1 Mitä tietokonenäkö on? 1
2.2 Tietokonenäön käyttömahdollisuudet 2

3 Kuvankäsittely 3

3.1 AForge.NET-kuvankäsittelykirjasto 3
3.2 Binaarinen iso objekti (BLOB) 4
3.3 Liitäntäkomponenttien merkintäalgoritmi 4
3.3.1 Yksisyöttöiset metodit 5
3.3.2 Kaksisyöttöiset metodit 5
3.3.3 Monisyöttöiset metodit 6

4 OCR 6

4.1 Tesseract OCR 7
4.2 Tesseractin toimintaperusteet 8

5 Projekti 8

5.1 Ohjelman tarve 9
5.2 Ohjelman toiminta 10
5.3 Rakenne 10
5.3.1 Visual Studio projektit 12
5.3.2 Tietokanta 13
5.3.3 Kuvasäiliö 14
5.4 Valitut työkalut 14
5.4.1 .NET-kehys 15
5.4.2 Entity Framework -kirjasto 15
5.4.3 Tesseract-kääre 16
5.5 Toteutus 16
5.5.1 Json-datan jäsentäminen rajapinnasta tietokantaan 17
5.5.2 Pdf-tiedoston tekstin louhinta ja kuvamuunnos 18
5.5.3 Levityskuvan reunavärin etsiminen 19
5.5.4 Pakkauksen sivujen leikkaaminen ja tallentaminen 20

5.5.5 Sivukuvan sijainti pakkauksessa 22
5.5.6 Sivukuvien suuntautumisen selvittäminen 23
5.6 Käyttöliittymä 24
5.7 Testiympäristö 27
5.8 Projektin aikana kohdatut ongelmat 28
5.8.1 Pdf-kirjasto 28
5.8.2 Vialliset materiaalit 29
5.8.3 Testiympäristön toimintaan saaminen 29
6 Jatkokehitys 30
7 Yhteenveto 31
Lähteet 32

Lyhenteet

2D	2-Dimensional. Kaksiulotteinen.
3D	3-Dimensional. Kolmiulotteinen.
AJAX	Asynchronous Javascript And XML. Joukko web-sovelluskehityksen tekniikoita.
API	Application Programming Interface. Rajapinta, josta voidaan hakea dataa kyselyiden avulla.
ASCII	American Standard Code for Information Interchange. Tietokonemerkistö, joka sisältää kirjaimille, numeroille sekä väli- ja erikoismerkeille numeeriset koodit.
BLOB	Binary Large Object. Binaarinen iso objekti on tietotyyppi, jolla kuvataan tietotyyppiä joka ei ole standardimuotoinen kuten esimerkiksi numero tai päivämäärä.
hOCR	Avoin tietojenkäsittelyn standardi optisesta merkkientunnistuksesta saadulle tekstille.
JSON	JavaScript Object Notation. Kevyt tiedonsiirtostandardi.
MVC	Model-View-Controller. Ohjelmistoarkkitehtuurityyli, joka erottelee käyttöliittymän toiminnallisten osat toisistaan.
OCR	Optical Character Recognition. Optinen merkkientunnistus.
ORM	Object-Relation Mapper. Tekniikka, jonka avulla voidaan kuvata oliomalli relaatiomallin mukaiseksi esitykseksi.
PDF	Portable Document Format. Tiedostomuoto.
PNG	Portable Network Graphics. Kuvan tallennusformaatti.

SPA	Single-Page Application. Yksisivuinen verkkosivusto, joka päivittää vain osan verkkosivusta.
TSV	Tab-Separated Values. Yksinkertainen tekstitiedostomuoto.
VNR	Varenummer. Yksilöivä tunnus lääkepakkauksille.
XML	Extensible Markup Language. Merkintäkieli.

1 Johdanto

Tämän insinöörityön tavoitteena on tehdä sovellus, joka automatisoi lääkkeiden ulkopakkausmateriaalien käsittelyprosessia ja helpottaa valmisteiden pakkauskuvien jakelua yrityksen asiakkaille. Automatisoinnilla pyritään minimoimaan henkilöstön käyttämää aikaa käsittelyprosessiin ja tekemään prosessista sujuvampaa sekä tehokkaampaa. Sovelluksen on tarkoitus tunnistaa ja irrottaa lääkepakkauksen sivut lääkeyritysten olemassa olevista materiaaleista. Materiaaleissa lääkepakkauksen sivut on rajattu yksivärisellä reunaviivalla, jonka perusteella kehitettävä ohjelma tunnistaa ja tallentaa pakkauksen sivut kuvina. Ohjelman tarkoituksena on myös pystyä määrittelemään lopputuloksena syntyvien ulkopakkauksen sivuista syntyneiden kuvien suuntautuminen sekä niiden suhteet toisiinsa.

Insinöörityö on jaettu kahteen osioon. Ensimmäisessä osiossa käsitellään projektiin liittyvää teoriaa, jonka jälkeen seuraavassa osiossa käsitellään projektin tekoa ja rakennetta. Teoriaosuus jakaantuu kertomaan tietokonenäön keskeisimmistä asioista ja projektiin käytetyistä komponenteista sekä niiden toiminnasta.

2 Tietokonenäkö

Ihminen pystyy henkilöiden ryhmäkuvaa tarkastellessaan laskemaan helposti henkilöiden lukumäärän ja tunnistamaan heidän mielentilojaan ilmeiden perusteella. Kuvasta voidaan tunnistaa ihmiset nimeltä ja todeta heidän välisiä suhteita. Tietokonenäön avulla pystytään saavuttaman vastaavia asioita useiden matemaattisten tekniikoiden avulla. Tekniikoilla pystytään luomaan osittaisia 3D-malleja useista päällekkäisistä valokuvista, joiden avulla voimme tunnistaa muun muassa valokuvista ihmisten kasvoja ja seurata videolla liikkuvaa kohdetta. (1.)

2.1 Mitä tietokonenäkö on?

Tietokonenäöllä pyritään jäljittelemään ihmisen näköaistia. Sen tarkoituksena on uudelleenrakentaa ja tulkita luonnollisia kohtauksia, jotka perustuvat erilaisten kameroiden ottamien kuvien sisältöön. Tietokonenäöllä on suuri rooli järjestelmissä, jotka sisältävät

valvontasatelliitteja, robotisoituja navigaatiojärjestelmiä, älykkäitä skannereita ja kaukokartoitusjärjestelmiä. Tällaiset järjestelmät sisältävät yleensä kameran tai muun vastaavan laitteen, joka lähettää visuaalista havaintoa tietokoneelle. Tietokonenäkö tutkii kuvan rakenteita ja siihen muodostuneita kuvioita. Tämän perusteella sitä voidaan pitää kuvankäsittelyn ja fotoniikaan ylempänä tasona. (2, s. vii)

Kuvankäsittely on digitaalisen kuvan muodostumisen ja manipulaation tutkimista. Kuvankäsittelyn avulla voidaan suodattaa kuvaa eri suodattimilla ja muuntaa kuvaa, sekä kuvasta voidaan poimia elementtejä, jotka ovat tarpeellisia käyttäjälle. Fotoniikka on tietokonenäön kuvakulmasta valotiedettä, jolla kaapataan visuaalisia kohtauksia. (2, s. vii)

2.2 Tietokonenäön käyttömahdollisuudet

Tietokonenäköön pohjautuvia järjestelmiä käytetään nykyään laajasti monissa erilaisissa sovelluksissa ja ohjelmissa. Tietokonenäköön liittyvät tekniikat ovat edistyneet merkittävästi viime vuosikymmenillä, ja sen avulla voidaan tuoda erilaisiin järjestelmiin muun muassa turvallisuutta parantavia tekijöitä. Tietokonenäköä käytetään apuna muun muassa

- tekstintunnistuksessa
- sormenjälki- ja kasvontunnistuksessa
- turvallisuus- ja valvontalaitteissa
- 3D-mallinnuksessa
- koneellisessa laaduntarkistuksessa.

Tekstintunnistuksen avulla pystytään laajasti lukemaan eri paikoissa esiintyvää tekstiä, esimerkiksi käsin kirjoitettuja kirjeitä ja autojen rekisterikilpiä (ks. luku 4). Tekstintunnistuksessa käytettyä tekniikkaa käytetään myös sormenjälkien ja kasvojen tunnistuksessa. Sormenpään kuvioinnista ja kasvonpiirteistä etsitään samankaltaisuuksia vertailtavaan kohteeseen ja päätellään, ovatko kuviot tarpeeksi identtiset ollakseen keskenään samoja. (1.)

Turvallisuus- ja valvontalaitteissa tietokonenäön avulla pystytään tarkastelemaan ympäristöä ja seuraamaan ympärillä olevia objekteja. Objektit voivat olla esimerkiksi ihmisiä, eläimiä tai kulkuneuvoja. Esimerkiksi nykypäivänä kulkuneuvoihin, kuten henkilöautoihin, on kehitetty järjestelmä, joka soveltaa tietokonenäköä. Järjestelmän tehtävänä on

havaita odottamattomia esteitä ja yrittää välttämään henkilövahinkoja, kuten auton eteen syöksyviä jalankulkijoita. Järjestelmän tarkoituksena on tunnistaa henkilöauton eteen tuleva objekti ja sen perusteella pystyä arvioimaan tehtävä toimenpide. Järjestelmän tarkoituksena on täydentää aktiivisten näkötekniikoiden, kuten tutkan toimintaa. (1.)

Yksittäisestä tai useammasta kuvasta, jossa esiintyy jokin objekti, pystytään luomaan 3D-malli. Esimerkiksi talosta otetaan kuvia eri puolilta, jonka jälkeen tietokonenäön avulla siitä pystytään rakentamaan kolmiulotteinen mallikuva. Mallikuva pystytään luomaan hyödyntämällä tietoa siitä, että talo koostuu suurista tasoista ja muista geometrisista muodoista, jotka suuntautuvat kohtisuoraan painovoimaa ja muita muotoja vastaan. (1.)

Koneellisen laaduntarkastuksen avulla pystytään esimerkiksi nopeasti etsimään vikoja teräsvalukappaleista käyttäen röntgensäteilyä. Tietokonenäkö pystyy havaitsemaan ihmissilmää tarkemmin vikoja kappaleesta. (1.)

3 Kuvankäsittely

Nykypäivänä ihmiset kuvaavat ympäristöään erilaisilla kameroilla, jotka sisältävät automaattisia ominaisuuksia, joilla pystytään analysoimaan otettuja kuvia. Näihin sovelluksiin liittyy erilaisia kuvankäsittelyalgoritmeja, joilla kuvasta poimitaan ominaisuuksia, jotka mahdollistavat kuvan analysoinnin. Tällaisia algoritmeja on kehitetty muun muassa ominaisuuksien ja reunaviivojen etsimiselle, kuvan suodattamiseen ja kontrastin parantamiselle, kuvan segmentoinnille sekä kuvan kompressoimiseen (3). Tässä luvussa perehdymme .NET-pohjaiseen AForge-kuvankäsittelykirjastoon ja sen toimintaperiaatteisiin.

3.1 AForge.NET-kuvankäsittelykirjasto

AForge.NET on vapaan lähdekoodin kehys, joka on suunniteltu kehittäjille, jotka haluavat hyödyntää tietokonenäköä ja keinoälyä tuotoksissaan. Kehys koostuu useista kirjastoista, joita voidaan hyödyntää muun muassa kuvankäsittelyssä, hermoverkoissa ja koneoppimisessa. (4.) AForgen sisältämistä kirjastoista tässä projektissa käytetään vain AForge.Imaging- ja AForge.Math-kirjastoja. AForge.Imaging-kirjasto on kehyksen sisältämistä kirjastoista suurin. Se sisältää kuvankäsittelyyn liittyviä rutiineja ja suodattimia,

joiden tarkoituksen on auttaa kuvankäsittelyyn sekä tietokonenäköön liittyvissä tehtävissä. AForge.Math-kirjasto sisältää erilaisia matematiikkaan liittyviä algoritmeja, kuten SimpleShapeChecker-luokan, jolla voidaan tunnistaa geometrisia kuvioita. (5.)

AForge tarjoaa BlobCounter-luokan, joka kerää kaikki kuvassa esiintyvät blobit, eli objektit (katso tarkemmin luku 3.2) listaan. BlobCounterille voidaan antaa asetuksia, joiden perusteella se suodattaa tuloksista tietyt objektit pois. Sille voidaan antaa esimerkiksi etsittävien objektien minimi- ja maksimikorkeus sekä leveys. Tällöin se suodattaa tuloksista pois kaikki ne, jotka eivät täytä ehtoja. BlobCounterille voidaan myös kertoa, mihin järjestykseen se laittaa löytyneet objektit. Objektit voidaan järjestellä suurimmasta pienimpään ja pienimmästä suurimpaan, tai löytyneistä objekteista voidaan valita pelkästään suurin. (6.)

BlobCounter-luokan avulla löytyvien objektien prosessointi tehdään liitäntäkomponenttien merkintäalgoritmia (Connected Components Labeling) käyttämällä. BlobCounterin käyttämä algoritmi on lähdekoodin perusteella yksisyötteinen (7) ja sen objektipikselien tunnistus tapahtuu 8-liitettävyyttä käyttäen (8). Liitäntäkomponenttien merkintäalgoritmista selitetään tarkemmin luvussa 3.3.

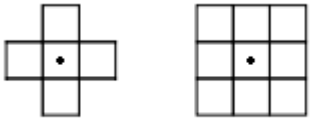
3.2 Binaarinen iso objekti (BLOB)

Binaarinen iso objekti eli Binary Large Object (BLOB) on tietotyyppi, johon voi tallentaa binaariobjekteja tai -tietoja. Binaarisella iso objektilla kuvataan tietotyyppiä, joka ei ole standardimuotoinen tietotyyppi kuten esimerkiksi numero tai päivämäärä. Näitä käytetään muun muassa tietokannoissa kuvien ja videoiden tallentamiseen. (9.)

3.3 Liitäntäkomponenttien merkintäalgoritmi

Liitäntäkomponenttien merkintäalgoritmi on menetelmä, jolla määritellään yksikäsitteinen tunniste jokaiselle binaarikuvassa olevalle objektille tai siihen liitettyyn komponenttiin. Tunnisteen määritys tapahtuu niin, että jokaisella objektin pikselillä on sama tunniste. Binaarikuva sisältää kahdenlaisia pikseleitä: objektipikseleitä ja taustapikseleitä. Objektipikselit ovat objektin muodostavan kuvion pikseleitä ja taustapikselit ovat kuvion ympärillä olevia pikseleitä. (10.)

Objektien tunnisteiden muodostaminen 2D-kuvaan voidaan tehdä joko 4- tai 8-liitännällä (10). Kun käytetään 4-liitäntää, lasketaan naapureiksi vain pikselit suoraan ylhäällä, alla, oikealla ja vasemmalla. Kun taas 8-liitännässä neljän vierekkäisen pikselin lisäksi lasketaan myös diagonaaliset pikselit naapureiksi. (11.)



Kuva 1. Vasemmalla 4-liitettävyys ja oikealla 8-liitettävyys.

Liitäntäkomponenttien merkintäalgoritmi on välttämätön osa useimpia sovelluksia, joissa käytetään kuvantunnistusta tai tietokonenäköä, sillä sen palauttamat tunnisteet ovat molemmissa avainasemassa. Näiden tunnisteiden avulla pystytään esimerkiksi tunnistamaan erilaisia merkkejä ja kuvioita tunnistettavasta kuvasta. (10.)

Tunnisteiden löytämiseksi on kehitetty erilaisia tapoja. Merkintäalgoritmi käy kuvan pikseleitä läpi eri metodien avulla. Nämä metodit ovat yksisyöttöinen, kaksisyöttöinen ja monisyöttöinen. Metodeista nopein on yksisyöttöinen ja seuraavaksi kaksisyöttöinen. Nopeus riippuu täysin siitä, kuinka monta iteraatiota metodin aikana tehdään kuvassa oleville pikseleille. (10.)

3.3.1 Yksisyöttöiset metodit

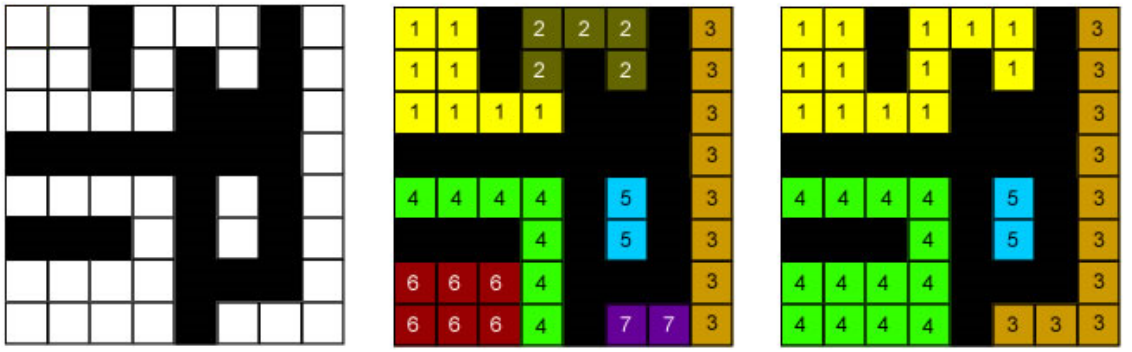
Liitäntäkomponenttien merkintäalgoritmin ollessa yksisyöttöinen, algoritmi skannaa sille annetun kuvan löytääkseen objektin merkitsemättömiä pikseleitä. Samalla algoritmi antaa jokaiselle objektiin yhdistetylle pikselille saman tunnisteen. Yksisyöttöiset algoritmit käyvät kuvan pikselit läpi vain kerran, mutta tyypillisesti epäsäännöllisen ja vaihtuvan kaavan mukaan. Yksisyöttöinen algoritmi on rekursiivinen, eli se kutsuu itseään algoritmin läpikäynnin aikana. Algoritmi on vaatimusluokaltaan lineaarinen. (10.)

3.3.2 Kaksisyöttöiset metodit

Kaksisyöttöinen liitäntäkomponenttien merkintäalgoritmi toimii kolmessa eri vaiheessa. Ensin on skannausvaihe, sitten analyysivaihe ja viimeiseksi merkintävaihe. Skannausvaiheessa kuva skannataan kerran ja jokaiselle objektipikselille määritetään väliaikainen

tunniste. Analyysivaiheessa analysoidaan tunnisteiden yhteneväisyyttä, jotta lopulliset tunnisteet pystytään määrittämään. Merkintävaiheessa objektipikseleille annetaan lopulliset tunnisteet, kun kuva skannataan uudestaan läpi ja saatuja tietoja verrataan edellisen ajokerran merkintöihin. (10.)

Kaksisyöttöinen algoritmi voidaan toteuttaa myös kaksivaiheisena, jolloin analyysivaihe voidaan integroida joko skannausvaiheeseen tai merkintävaiheeseen. Vaiheiden yhdistely riippuu siitä, miten tunnisteiden vastaavuustietojen rakenne on tehty datarakenteeseen. (10.)



Kuva 2. Objektipikseleiden tunnisteiden kehittyminen kaksisyöttöisen algoritmin aikana.

3.3.3 Monisyöttöiset metodit

Monisyöttöinen liitäntäkomponenttien merkintäalgoritmi toimii samalla tavalla kuin kaksisyöttöinen algoritmi. Monisyöttöisen algoritmin erona on se, että se käy kuvan useaan kertaan läpi ennen kuin lopulliset tunnisteet pystytään antamaan objektipikseleille. (10.)

4 OCR

OCR (Optical character recognition) eli optinen merkkien tunnistus on tekniikka, joka kääntää käsinkirjoitettua tai painettua tekstiä koneellisesti koodatuksi tekstiksi. Se on prosessi, jossa käsitellään kirjoitetut merkit ja tunnistetaan ne. (12, s. 1.) Prosessin aikana asiakirjan, kuvan tai minkä tahansa tiedoston teksti muutetaan ASCII-muotoisiksi

merkeiksi, joita tietokone pystyy lukemaan (13). ASCII (American Standard Code for Information Interchange) on merkkijono, joka korvaa kunkin kirjaimen, numeron ja erikoismerkin 7-bittisellä binäärisellä numerolla. Merkkijono koostuu tällöin seitsemästä numerosta, jotka ovat joko nollia tai ykkösiä. (14.)

OCR on yksi suosituimmista tutkimuskohteista kuviontunnistuksessa viime vuosikymmeninä. Se on aktiivisesti tutkittu aihe teollisuudessa ja korkeakouluissa, koska siinä on hyvää potentiaalia sovelluskehitykselle. OCR-tekniikkaa tutkittiin alun perin 1930-luvun alussa ja sen alkuperä sijaitsee Saksassa, jossa Gustav Tauschek aloitti sen kehityksen. (12, s. 1.)

4.1 Tesseract OCR

Tesseract on vapaan lähdekoodin OCR-moottori, jonka kehityksen on alun perin aloittanut Hewlett-Packard vuonna 1985. Vuonna 2005 HP julkaisi Tesseractin lähdekoodin, minkä jälkeen vuodesta 2006 eteenpäin sen kehitys on siirtynyt Googlelle. (15.)

Tesseract pystyy tunnistamaan yli 100 eri kieltä, ja se tukee useita eri formaatteja. Esimerkiksi se pystyy palauttamaan pelkkää tekstiä sekä pdf-, tsv- ja hocr-muotoisia tiedostoja. (15.) Kuvassa olevien merkkien tunnistus tapahtuu käyttämällä apuna Tesseractin mukana olevaa Leptonica-kirjastoa (16), jonka PIX-luokkaan voidaan syöttää tunnistettavaa tekstiä sisältävä kuva (17). Tesseractia voidaan käyttää sen mukana tulevan komentoriviltä toimivan käyttöliittymän kautta tai se voidaan liittää projektiin kirjastoksi. Tesseract toimii suoraan C- ja C++-kielisissä ohjelmissa, mutta muihin kieliin se tarvitsee kääreen toimiakseen. (15.)

Tesseract ei aina pysty tuottamaan laadultaan hyvää tulosta. Se ei välttämättä tunnista kaikkia kuvassa esiintyviä kirjaimia tai edes kokonaisia osia tekstistä. Syitä tähän on useita. Esimerkiksi sille annetun kuvan laatu saattaa olla liian huono, kuva on vinossa, tai kuvassa käytetään fonttia tai kieltä, jota sille ei ole opetettu. Tesseract toimii parhaiten, kun kuvan resoluutio on vähintään 300. (16.) Käyttäjän halutessa Tesseract voidaan opettaa lukemaan uutta tekstin fonttia tai kieltä, jos se koetaan tarpeelliseksi (15). Tesseractia voidaan käyttää myös tunnistamaan asiakirjojen tai kuvien suuntaa; onko kuva oikeinpäin vai ei.

4.2 Tesseractin toimintaperusteet

Tesseract olettaa, että sille annettu kuva on binäärinen ja se sisältää tekstialueita. Sille annetun kuvan käsittely tapahtuu vaiheittain. Ensimmäisenä vaiheena on analyysi, jossa kuvan sisältämät ääriviivat tallennetaan blobeiksi, eli objekteiksi. Analyysi tehdään käyttämällä yhdistelmäkomponenttien merkintäalgoritmia. Objektit järjestetään tekstiviivoiksi, jotka analysoidaan suhteelliseksi tekstiksi. Tämän jälkeen tekstiviivat jaetaan sanoiksi. (18.)

Seuraavana vaiheena on kaksivaiheinen prosessi, jonka ensimmäisessä vaiheessa yritetään tunnistaa jokainen tekstissä esiintyvä kirjain. Jokainen kirjaimista muodostettu sana, joka on tarkkuustasoltaan tarpeeksi korkea, välitetään mukautuvan luokittelijalle opetusdataksi. Mukautuva luokittelija on yksikkö, joka oppii tekstin merkeistä ajon aikana ja kehittyy. Tämä antaa tekstissä jäljellä oleville kirjaimille ja sanoille mahdollisuuden tarkempaan tunnistukseen. Prosessin ensimmäisen vaiheen jälkeen suoritetaan toinen vaihe siltä varalta, että mukautuva luokittelija on oppinut sille syötetystä opetusdatasta liian myöhään jotain hyödyllistä. Vaiheen aikana tunnistetaan uudestaan sanoja, joita ei ole aiemmin tunnistettu hyvin. (18.)

5 Projekti

Insinöörityön tarkoituksena oli luoda ohjelma, joka pystyy sille annetun tiedoston perusteella poimimaan tiedostosta lääkepakkauksen kaikki sivut ja tallentamaan poimitut sivut omiksi kuviksi Azuren tuottamaan BlobStorage-pilvipalveluun. Ohjelmalle annettava tiedosto pitää sisällään muun muassa lääkepakkauksen mitat, painotiedot sekä levityskuvan, joka on rajattu vaihtelevalla värillä riippuen valmistajasta ja valmisteesta (kts. kuva 3). Ohjelma tunnistaa kuvasta pakkauksen ääriviivan värin, jonka avulla se leikkaa ulkopakkauksen sivukuvat kuviksi. Tämän jälkeen Tesseract-kirjaston avulla tarkistetaan kuvien suuntautuminen ja mittaillaan kuvien alkuperäisten reunapisteiden avulla kuvien suhteita toisiinsa nähden.

Kuva 3. Esimerkki pakkausmateriaaleista löytyvästä valmisteen levityskuvasta.

5.1 Ohjelman tarve

Lääketietokeskuksen asiakkailla on tarve saada keskitetysti ajantasaiset lääkkeiden ulkopakkausten kuvat sekä mitat esimerkiksi verkkosivuja, varaston hallintaa tai 3D-malleja varten. Erilaisia lääkepakkauksia on markkinoilla Suomessa noin 10 000 ja niitä päivitetään usein. Lääkeyrityksillä on valmiina pakkauksia koskeva dokumentaatio, jonka hyödyntäminen on tehokkaampaa kuin toistuvasti muuttuvien yksittäisten lääkepakkausten valokuvaaminen. Materiaaleja hyödyntämällä pystytään tarjoamaan Lääketietokeskukselle sekä yrityksen asiakkaille ajantasaista tietoa lääkepakkauksen ulkomuodoista ja sen mitoista. Lääkepakkausten ulkomitoille on ollut kysyntää, sillä niiden perusteella pystytään muun muassa määrittämään, kuinka paljon yksi pakkaus vie varastointitilaa.

5.2 Ohjelman toiminta

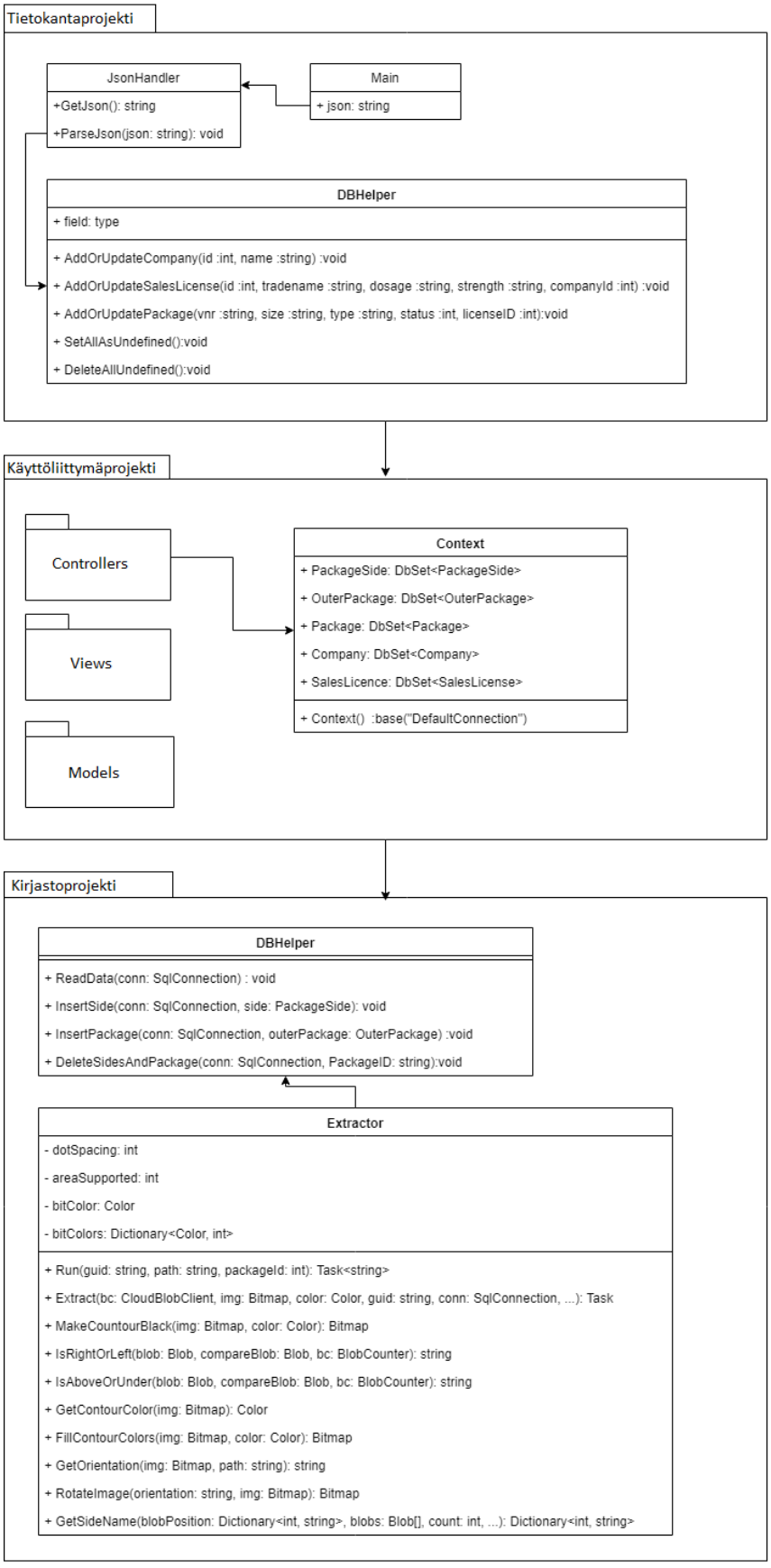
Ohjelman toiminta jakaantuu useaan eri osaan, joissa se työstää sille annetun tiedoston haluttuun muotoon. Ohjelmalle annettu lähdetiedosto voi olla joko pdf- tai png-muotoinen. Ensimmäinen vaiheista on tiedoston muuntaminen kuvaksi, jos lähdetiedosto on pdf-muotoinen. Seuraavaksi ohjelma etsii kuvasta pikselien värien avulla levityskuvan reunavärin sekä etsii kuvasta suorakulmion muotoisia objekteja AForge-kirjaston avulla ja poimii vain levityskuvaan liittyvät suorakulmiot. Suorakulmiot poimitaan tämän jälkeen ja irrotetaan omiksi kuvatiedostoiksi, minkä jälkeen kuvatiedostojen suuntautuminen tarkistetaan tekstintunnistuskirjaston avulla.

Ohjelma käynnistyy, kun käyttäjä syöttää järjestelmään tiedoston. Tiedosto ladataan Azuren BlobStorage-pilvipalvelun säiliöön, josta ohjelma käy hakemassa sinne tuodut tiedostot. Ohjelma tarkistaa ensimmäisenä sille annetun tiedoston muodon ja sen perusteella päättää siirtyykö se suorittamaan kuvamuunnosta syötetylle tiedostolle. Jos ohjelmalle annettu tiedosto on jo kuvaformaattia, jättää ohjelma kuvamuunnoksen väliin ja siirtyy seuraavaan vaiheeseen.

Kuvamuunnoksen jälkeen kuvasta etsitään lääkepakkauksen reunaviivan väri ja kaikki reunaviivan väriä vastaavat pikselit muutetaan mustiksi. Tämän jälkeen AForge-kirjaston avulla kuvasta etsitään suorakulmioita, jotka tallennetaan muistiin. Seuraavassa vaiheessa suorakulmiot käydään läpi ja niiden vierekkäisyyttä tarkastellaan verrattuna isoimpaan löydettyyn suorakulmion muotoiseen kuvaan. Ulkopakkauksen etukuva on yleensä pakkauksen isoin kuva, ja tämän takia isoin löytynyt suorakulmion muotoinen kuva valitaan aloituskuvaksi vertailuun. Toistensa vierekkäiset kuvat, jotka ovat aloituskuvan kanssa kytköksissä tallennetaan pilveen. Ne kuvat, joilla ei ole vierustovereita isoimpaan kuvaan liittyen, hylätään ylimääräisinä.

5.3 Rakenne

Ohjelma koostuu kolmesta eri Visual Studion projektista, tietokannasta ja kuvasäiliöstä. Projektit hakevat ja tallentavat tietoa tietokantaan ja ohjelman lopputuloksena kuvasäiliöön tallennetaan pakkauksen sivukuvat.



Kuva 4. Luokkakaavio projektien luokista ja projektien kytköksistä toisiinsa.

5.3.1 Visual Studio projektit

Ohjelma koostuu kolmesta eri Visual Studio -projektista, jotka on niputettu yhdeksi Visual Studion ratkaisuksi (solution). Ensimmäinen projekti on kirjastoprojekti, jossa on ohjelman koko logiikka. Toinen projekti populoi tietokannan Vnr-apista saadulla json-muotoisella lääketiedolla. Kolmas projekti on käyttöliittymä, joka tekee metodikutsuja kirjastoprojektiin ja näyttää tietokannassa olevia tietoja käyttäjälle. Kirjastoprojekti on itsenäinen, eikä se käytä muita projekteja. Tietokantaprojekti käyttää käyttöliittymäprojektin malliluokkia hyödykseen ja käyttöliittymäprojekti käyttää kirjastoprojektin funktioita, kun ohjelmaan tahdotaan luoda uusi ulkopakkaus valmisteelle.

Vnr-api on rajapinta, josta pystytään hakemaan dataa http-pyynnöllä, johon on liitetty kirjautumiseen käytetty käyttäjä-salasana-pari. Rajapinnasta saatu data sisältää muun muassa tietoa valmisteiden pakkausten vnr-numeroista, lääkeaineiden vahvuuksista sekä eri valmisteiden markkinoijista. Saadusta datasta vain murto-osa jäsennetään projektin käyttämään tietokantaan, sillä suurin osa saadusta datasta on tarpeetonta projektin suhteen.

Vnr-numero on lyhenne norjankielisestä sanasta varenummer, joka tarkoittaa tuotenumeroa. Vnr-numero on kuusinumeroinen yksilöivä tunnus lääkepakkaukselle, ja se on numero väliltä 000001-199999 tai 370000-599999. Sillä pystytään tunnistamaan yksittäisiä lääkepakkauksia luotettavasti koko lääkkeen jakeluketjun ajan. Vnr-numero annetaan kaikille ihmisille tarkoitetuille lääkkeille, eläinlääkkeille, kasviperäisille lääkkeille sekä perinteisille kasvirohdosvalmisteille. (19.)

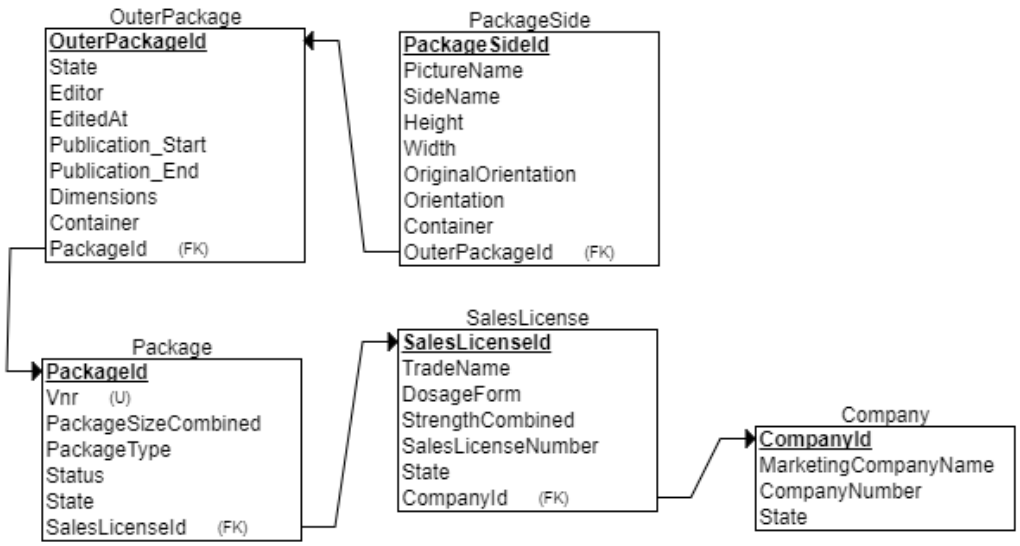
API (Application Programming Interface) eli rajapinta on joukko rutiineja, protokollia ja muita työkaluja, joiden avulla pystytään määrittelemään, miten sovelluksen komponentit kommunikoivat keskenään (20). Rajapintaa voidaan kuvitella grillikioskina, jossa kioski toimii tietojärjestelmänä, josta haetaan tietoa rajapinnan kautta. Kioskin myyjä on rajapinta, jolta asiakas käy kyselemässä ruokaa. Asiakas ei tiedä täysin, mitä kioski sisältää, mutta tietää ulkona olevan ruokalistan perusteella millä kyselyllä myyjältä saa ostettua esimerkiksi makkaraperunat. Grillikioski toimii esimerkkinä rajapinnasta, sillä asiakas ei pääse itse kioskiin sisään, vaan joutuu luukun kautta käymään keskustelua myyjän kanssa saadakseen haluamansa.

Rajapinnat voidaan jakaa kahteen ryhmään: datarajapintoihin ja toiminnallisiin rajapintoihin. Datarajapinnoista saadaan noudettua kyselyillä dataa, tästä esimerkkinä aiemmin

mainittu vnr-api. Toiminnalliset rajapinnat ovat taas rajapintoja, joiden kautta pystytään esimerkiksi tallentamaan tietoa tietorakenteisiin. Rajapinnat voivat olla myös avoimia, jolloin niiden dokumentaatio on julkaistu kaikkien nähtäville. Tämä ei kuitenkaan tarkoita sitä, että kaikki pääsevät niihin suoraan käsiksi. Rajapintaan pääsyä on voitu rajoittaa käyttöoikeuksilla, jotka saadaan ostamalla esimerkiksi kuukauden lisenssi rajapintaan. (20.) Aiemmin kerrottu esimerkki grillikioskista on hyvä esimerkki avoimesta rajapinnasta. Tuotevalikoima, joka esitellään kioskin seinällä vastaa dokumentaatiota, joka on kaikkien saatavilla. Kuitenkaan tilattua makkaraperuna-annosta ei saada, ellei siitä makseta tiettyä summaa myyjälle.

5.3.2 Tietokanta

Ohjelmassa käytetty tietokanta on Azuren pilvipalveluissa toimiva Microsoft Azure SQL -tietokanta. Tietokanta sisältää viisi taulua, jotka ovat kytköksissä toisiinsa yhdestä moneen suhteilla. Tietokannan kolme taulua populoidaan rajapinnasta saadulla json-muotoisella datalla. Näitä kolmea taulua voidaan kutsua ohjelman perustiedoiksi. Tietokannassa olevat kaksi muuta taulua täyttyvät, kun järjestelmään lisätään valmisteille ulkopakkauksia. Näitä tietoja voidaan kutsua järjestelmän omiksi tiedoiksi.



Kuva 5. Tietokannan rakenne.

Tietokanta on toteutettu code-first-lähestymistavalla, eli kirjoitettu ohjelmakoodi hallitsee tietokannan rakennetta. Koodin ja tietokannan välille on asetettu migraatio, joka pitää huolen tietokannan viittauksista toisiinsa. Tämä estää muun muassa datan tallentamista väärään paikkaan, jos tietokantakoodia on muokattu. Tällaisia tilanteita olisivat esimerkiksi tietokannan malliluokkien muuntaminen, lisääminen ja poistaminen. (21.)

Tietokannan migraatio on tehty ajamalla Visual Studion pakettien hallintaan käytettävällä komentorivillä komennot

- Enable-Migrations
- Add-Migration
- Update-Database.

Komento Enable-Migrations hyväksyy migraation lisäämisen projektiin ja luo Migrations-kansion projektiin. Kansio sisältää konfiguraatioluokan ja migraatiotiedoston. Konfiguraatioluokan kautta käyttäjä pystyy määrittämään, miten migraatio käyttäytyy tietokanta-kontekstissa. Migraatioluokka sisältää tiedon tietokantaan luoduista tietokantatauluista. Add-Migration-komento lisää uuden migraation malliluokkiin tehtyjen muutosten perusteella ja Update-Database-komento hyväksyy ja päivittää tietokantaan vireillä olevien migraatioiden muutokset. (21.)

5.3.3 Kuvasäiliö

Projektin lopputuloksena syntyvät lääkepakkauksien sivut tallennetaan Azuren tuottamaan pilvessä sijaitsevaan säiliöön (blob storage). Säiliö sisältää kolme eri alasäiliötä: yksi käyttäjän lataamille tiedostoille, toinen syntyville lopputuloksille ja kolmas säiliö tiedostoille, joita ei pystytä käsittelemään ohjelman avulla.

5.4 Valitut työkalut

Projektin alkuvaiheessa ei ollut vielä päätetty, millä ohjelmointikielellä ohjelmaa lähdetään rakentamaan. Vaihtoehtoina olivat Java sekä C#. Päädyin lopulta valitsemaan ohjelmointikieleksi C#:n, koska halusin opetella uutta kieltä. Kieliä vertaillessa totesin, että kielet ovat melko samanlaisia, muun muassa niiden syntaksin osalta. C#-kielessä on muutamia eroavaisuuksia, kuten esimerkiksi using-metodi, jolla pystytään muun muassa luomaan jokin olio vain tietyksi ajaksi muistiin. Toisena syynä kielen valitsemiseen oli se,

että C#:lle tuntui löytyvän paljon enemmän kirjastoja, joita pystyi käyttämään projektissa hyväksi. Myös Lääketietokeskuksen sisäinen osaaminen oli yksi syy valita C#-projektiin.

5.4.1 .NET-kehys

Käyttöliittymän tekemiseen valittiin .NET Framework Core 2.0 -kehys, koska sen teknologia on modernia ja se on huomattavasti kevyempi ja nopeampi kuin edelliset versiot .NET Framework -kehyksestä. Kehys ei kuitenkaan osoittautunut yhteensopivaksi muiden kirjastojen kanssa, ja se jouduttiin vaihtamaan vanhempaan .NET Framework v4.6.1 -kehykseen, joka on huomattavasti hitaampi verrattuna Core 2.0 -kehykseen (22). Hitauden pystyi huomaamaan, kun pyöritti ensin Core-kehykseen tehtyä käyttöliittymän alkua ja sen jälkeen samaa käyttöliittymää Framework 4.6.1 -kehyksen kautta. Framework 4.6.1 -kehyksellä kesti huomattavasti kauemmin toteuttaa käyttäjän tekemä pyyntö. Siirto toiseen näkymään kesti useamman sekunnin, kun taas Core-kehystä käyttäen käyttäjä siirrettiin jouhevasti silmän räpäyksessä uuteen näkymään.

5.4.2 Entity Framework -kirjasto

Tietokantaprojektin ja tietokannan välinen yhteydenpito, sekä käyttöliittymän tekemät tietokantakyselyt hoidetaan Entity Frameworkin kautta. Entity Framework on .NET-ympäristölle kehitetty ORM-kirjasto (Object-Relation Mapper), joka vähentää relaatiotietokantojen ja objektien viittausten välistä epäsuhtaa. Entity Frameworkin avulla relaatiotietokantaan tallennettuihin tietoihin pystyy vuorovaikuttamaan käyttämällä voimakkaasti määriteltyjä objekteja. (23.) ORM on tekniikka, jonka avulla voidaan käsitellä dataa objektien avulla, kartoittamalla ohjelman objektit yhteensopiviksi dataan, joka sijaitsee relaatiotietokannassa (24).

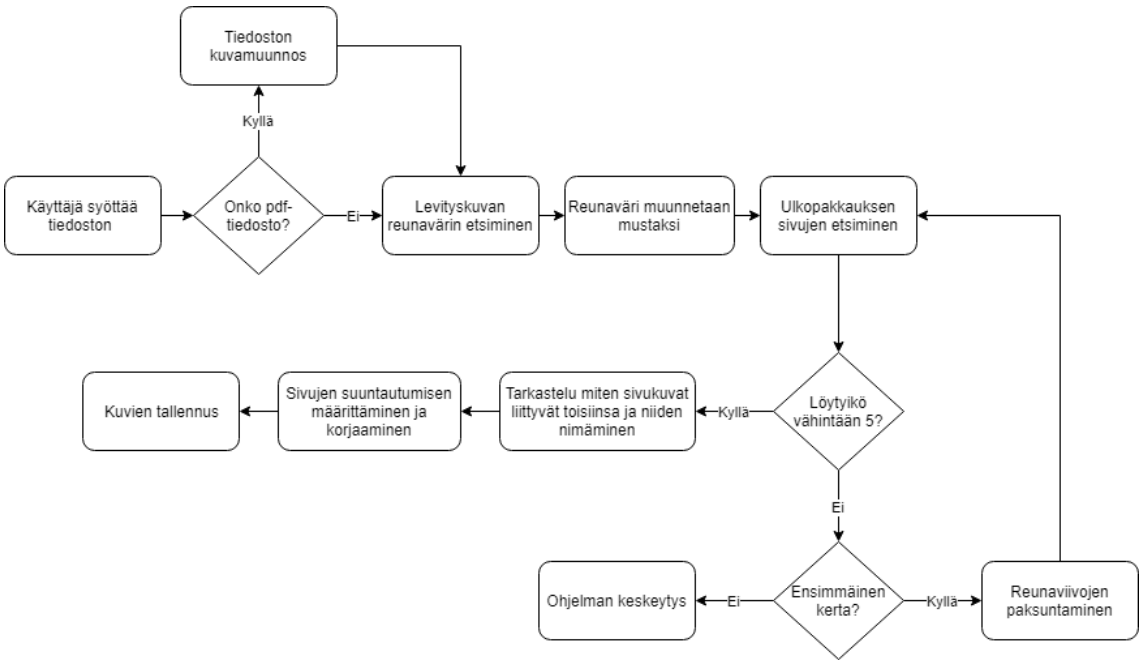
Toisena vaihtoehtona tietokannan ja projektien väliseen yhteydenpitoon olisi ollut Hibernate ORM -kirjasto. Entity Framework valittiin projektiin tekemättä sen suurempaa vertailua, sillä Entity Framework oli rakennettu .NET-pohjaisille sovelluksille eikä sen integroiminen projektiin vaatinut paljoa. Entity Framework oli suoraan käytettävissä latauksen jälkeen, ja sen konfiguroiminen käyttämään tehtyä tietokantaa oli helppo. Tietokannan määrittäminen Entity Frameworkille vaati yhteysavaimen konfiguroimisen web.config-tiedostoon ja tietokantakontekstiluokan luomisen. Kontekstiluokkaan määritettiin tietokannan malliluokat ja mitä yhteysavainta luokka käytti tietokantayhteyden luomiseen.

5.4.3 Tesseract-kääre

Sivukuvien suunnan määrittelemiseksi projektiin valittiin Charles Weldin tekemä .NET-kehykselle tarkoitettu kääre Tesseract-moottorista. Kääre käyttää Googlen tesseract-ocr -moottorin versiota 3.04 (25). Kääreestä on olemassa uudempi versio, joka käyttää Tesseractin uudempaa versiota, mutta uudemmalla versiolla Tesseractista varustettu kääre on vielä alpha-vaiheessa ja sitä kehitetään vielä (26). Tämän takia projektiin valittiin vanhempi versio OCR-moottorista, jonka versio on vakaa, eikä se ole enää kehitysvaiheessa. Tesseractin käyttö projektissa on melko pienessä osassa, eikä vaadi uudemman moottorin paranneltuja ominaisuuksia. Tesseract valittiin projektiin sen helppokäyttöisyyden takia, sekä sen takia, että se on yksi tarkimmista OCR-moottoreista, joka on julkaistu vapaana lähdekoodina (27).

5.5 Toteutus

Ohjelman logiikka on toteutettu kirjastoprojektiin, jossa on kolme luokkaa, joihin sovelluksen toiminta on rakennettu funktioina. Yksi luokka sisältää funktiot tietokantakyselyihin, toinen kaiken kuvien irrottamiseen liittyvän logiikan ja viimeinen luokka on pakkauksen yhden sivun malliluokka. Malliluokan tarkoituksena on helpottaa useiden samanlaisten sivukuvien tallennusta.



Kuva 6. Ohjelman kulkukaavio.

Tietokannan perustietojen tallennus tietokantaan tapahtuu erillisen VS-projektin kautta, joka hakee json-muotoisen datan rajapinnasta ja jäsentää sen tietokantaan. Näitä tietoja tarvitaan ohjelman suorituksen aikana, jotta ulkopakkaus osataan tallentaa oikealle lääkevalmisteelle, joka on oikean yrityksen valmistama. Samaa lääkevalmistetta voidaan myydä samalla kauppanimellä usean eri yrityksen toimesta.

5.5.1 Json-datan jäsentäminen rajapinnasta tietokantaan

Json-data haetaan rajapinnasta HttpWebRequest-kyselyn avulla. Kyselyn vastaus saadaan HttpWebResponse-kyselyn avulla, joka palauttaa HttpWebRequest-kyselyn vastauksen. Vastaus luetaan StreamReader-olion avulla muuttujaan, joka jäsennetään tietokantaan käyttäen Newtonsoftin tekemää Json.NET-kirjastoa. Kyselyn palauttama Json-data koostuu useista TradeName-elementeistä, jotka kostuvat useasta SalesLicenses-listasta, jossa on kaikki yrityksen omistamat lääkevalmisteet. Esimerkissä 1 on havainnollistettu lyhyesti json-datan rakennetta.

```
[{
      "TradeNames": [{
            "SalesLicenses": [{
                  "SalesLicenseID": "SalesLicenseID",
                  "License": "Licence",
                  "StrengthCombined": "40 mg ",
                  "DosageForm": "DosageForm",

                  "VNRs": [{
                        VNR": "VNR",
                        "PackageSizeCombined": "50 x 1,7 ml",
                        "TradeId": "TradeId",
                        "TradeName": "TradeName"
                  }]
            }]

            "CompanyId": "CompanyId",
            "MarketingCompanyName": "MarketingCompanyName"
      }]
}]
```

Esimerkkikoodi 1. Muokattu ja typistetty versio rajapinnasta saadusta datan rakenteesta.

Kun json-data on tallennettu muuttujaan, annetaan muuttuja json.NET-kirjaston JArray-elementtiin, jonka jälkeen muuttujan sisältämä data jäsennetetään. JArray-elementissä olevista objekteista saadaan foreach-rakenteen avulla poimittua tietyt objektit kutsumalla kirjaston SelectToken-metodia. Metodi etsii foreach-rakenteen vuorossa olevan elementin sisältämät objektit ja etsii metodille annettua string-muotoista tekstiä objektin nimestä. Esimerkkikoodissa 2 havainnollistetaan osittain edellisen json-muotoisen datan jäsentämistä json.NET-kirjaston avulla.

```
JArray array = JArray.Parse(json);

 foreach (JObject item in array) {
      string companyName = (string)item.SelectToken("MarketingCompanyName");
      int companyId = (int)item.SelectToken("CompanyId");
      var list = item.GetValue("TradeNames");

      foreach (JToken items in list) {
            string name = (string)items.SelectToken("TradeName");
            var salesLicenses = items.SelectToken("SalesLicenses");
             JArray licenceList = (JArray)salesLicenses;

             foreach (JObject li in licenceList) {
                  int SalesLicenseID = (int)li.SelectToken("SalesLicenseID");
            }
      }
}
```

Esimerkkikoodi 2. Lyhyt esimerkki rajapinnasta saadun datan jäsentämiseen.

5.5.2 Pdf-tiedoston tekstin louhinta ja kuvamuunnos

Azureen tallennettu pdf-muotoinen tiedosto muunnetaan kuvaksi käyttämällä Web-Supergoon tekemää ABCpdf-kirjastoa, joka osaa suorittaa pdf-tiedostolle rasterisaation, eli kuvamuunnoksen. Rasterisaatiossa pdf-tiedoston vektorimuotoinen esitys muunnetaan rasteriksi, eli kuvaksi, joka muodostuu pikseleistä. Tämän jälkeen muodostuva kuva ei enää skaalaudu tarkasti eri kokoiseksi, toisin kuin alkuperäinen vektorimuotoinen esitys. (28.) Kuvamuunnos joudutaan tekemään kirjaston avulla, sillä C#-kieli ei tarjoa alkuperäisesti minkäänlaista tukea pdf-tiedostojen konvertoimiseen toiseen muotoon. Edellä mainittu pdf-kirjasto oli ainut testatuista kirjastoista, joka osasi muuntaa pdf-tiedoston värit oikein kuvatiedostoon.

Kirjastolla oli tarkoitus lukea annetusta tiedostosta pakkauksen mitat ja lääkevalmisteen vnr-numero. Tämä ei kuitenkaan onnistunut, sillä useissa tiedostoissa ei ollut merkittynä pakkauksen mittoja selkeästi tai niitä ei ollut ollenkaan. Suurimmassa osassa tapauksia mitat ja niihin viittaavat avainsanat olivat jaoteltu taulukossa eri sarakkeisiin, jolloin mitat ja avainsana päätyivät pdf-kirjaston tuottamassa tekstitiedostossa omille riveilleen. Tiedostoon syntyi dokumentista riippuen useita rivejä (200-300 riviä), joiden joukosta oli hankalaa lähteä etsimään mittoja, jos avainsana ja mitat eivät olleet samalla rivillä. Sama ongelma oli myös Vnr-numeroiden kanssa.

Pakkauksen vnr-numeron kohdalla todettiin, ettei sitä tarvitse hakea materiaaleista, sillä kaikki yrityksen tiedossa olevien pakkausten vnr-numerot on ajettu tietokantaan vnr-apista. Käyttäjän tarvitsee vain valita käyttöliittymästä oikea pakkaus, jolle tahtoo lisätä uuden ulkopakkauksen. Tällöin pakkauksella on suoraan kaikki oikeat tiedot, kuten vnr-

numero ja valmistaja. Mittoja vnr-apista saadusta datasta ei kuitenkaan löydy, vaan ne joudutaan syöttämään järjestelmään manuaalisesti.

5.5.3 Levityskuvan reunavärin etsiminen

Kuvamuunnoksen jälkeen määritetään kuvan korkeus ja leveys. Tämän jälkeen lähdetään etsimään 20 pikselin välein jokaiselta tiedoston reunasivulta ensimmäistä pikseliä, joka ei ole musta tai valkoinen. Rajaus mustaan ja valkoiseen on tehty katsomalla materiaaleja ja toteamalla, että reunaväri ei ole ikinä musta. Tiedostoissa on myös aina käytetty valkoista taustaväriä, jonka perusteella oletetaan, ettei reunaviiva ole valkoinen. Kun ensimmäinen kriteereitä vastaava pikseli löydetään, tallentuu pikselin värikoodi Dictionary-rakenteeseen. Samalla kun värikoodi tallennetaan muistiin, vaihdetaan riviä, jolta etsitään seuraavaa värikoodia. Ensimmäisen kerran värin löytyessä tallentuu värin koodi alkion avaimeksi ja numero yksi sen arvoksi. Jos värikoodi löytyy uudestaan, kasvatetaan sen arvoa yhdellä. Lopuksi, kun kaikki reunapikselit on käyty läpi, palautetaan värikoodi, jonka arvo on suurin listassa.

Dictionary <TKey, TValue> on geneerinen luokka, jonka rakenteeseen voidaan tallentaa avain-arvo-pareja. Jokainen lisäys rakenteeseen koostuu arvosta ja siihen liittyvästä avaimesta. Rakenteeseen lisättävien arvojen pitää olla samaa tyyppiä, kuin rakennetta luotaessa määritetyt arvot TKey ja TValue. Dictionary on toteutettu hajautustaulukkona, joten se ei hyväksy duplikaatteja avainten seassa, vaan jokaisen avaimen täytyy olla uniikki. Hajautustaulukon ansiosta arvon hakeminen avaimen avulla rakenteesta on hyvin nopeaa. (29.)

Ohjelma käy pikseleitä yksitellen läpi getPixel-metodin avulla. Tämä on hidas operaatio, sillä ohjelmalle annetut kuvat ovat suuria ja sisältävät tuhansia pikseleitä. GetPixel-metodin avulla poimitusta pikselistä saadaan kaivettua pikselin värikoodi ketjuttamalla metodin perään metodi ToArgb. ToArgb palauttaa Color-rakenteesta pikselin 32-bittisen ARGB-väriarvon muodossa AARRGGBB. Muodon ensimmäinen osa AA esittää alfakomponenttiarvoa, joka on esityksen merkitsevin tavu. Heksadesimaaliesityksessä alfakomponenttiarvoa kuvataan risuaidalla (#). Muodon toinen, kolmas ja neljäs tavu, joita edustavat RR, GG ja BB, ovat värikomponentit punaiset, vihreät ja siniset arvot. (30.)

```
for (int j = width - 1; j >= 0; j -= 20) {

      for (int i = height - 1; i >= 0; i--) {

            if (!Color.White.ToArgb().Equals(img.GetPixel(j, i).ToArgb()) &&
            !Color.Black.ToArgb().Equals( img.GetPixel(j, i).ToArgb())){

                  bitColor = img.GetPixel(j, i);

                  if (bitColors.Keys.Contains(bitColor)) {
                        bitColors[bitColor]++;
                  } else {
                        bitColors.Add(bitColor, 1);
                  }

                  break;
            }
      }
}
```

Esimerkkikoodi 3. Funktion osa, jossa haetaan kaikki tiedoston oikean reunan pikselien värit listaan käymällä rivejä läpi ylhäältä alas.

5.5.4 Pakkauksen sivujen leikkaaminen ja tallentaminen

Reunaviivan etsimisen jälkeen ohjelma etsii kuvasta kaikki värikoodia vastaavat pikselit ja muuttaa ne mustiksi setPixel-metodin avulla. Pikselien värimuunnos tehdään siksi, että AForge-kirjasto pystyy lukemaan kuvaa ja etsimään kuvasta suorakulmion muotoisia alueita. Ohjelma muuttaa myös hiukan eri sävyisiä pikseleitä mustaksi, koska reunaviivan värin sävy vaihtelee. Tämä johtuu pdf-tiedoston kuvamuunnoksesta, joka ei ole pystynyt tuottamaan tarkalleen samaa väriä koko reunaviivan matkalle. Sävyyn on jätetty 10 sävyasteen heittovara.

AForge-kirjaston etsittäväksi väriksi on määritetty oletusväriksi musta, jota pystyy muuttamaan kirjaston BackgroundThreshold-metodilla (31). Metodia ei voida kuitenkaan asettaa käyttämään etsittyä reunaväriä sävyn muutosten takia, vaan tämän takia reunaviivan pikselit joudutaan muuntamaan mustiksi.

AForge-kirjasto etsii BlobCounter-metodin avulla kuvasta kaikki määritetyn kokoiset kappaleet. Tämän jälkeen löydetyistä kappaleista etsitään oikean muotoiset kappaleet SimpleShapeCheck-metodin avulla, jonka jälkeen se tallentaa jokaisen löytyneen kappaleen reunapisteet listaan.

```
BlobCounter blobCounter = new BlobCounter();

blobCounter.FilterBlobs = true;
blobCounter.MinHeight = 110;
blobCounter.MinWidth = 110;
blobCounter.MaxHeight = 1700;
blobCounter.ProcessImage(image);
int size = 0;

Blob[] blobs = blobCounter.GetObjectsInformation();
SimpleShapeChecker shapeChecker = new SimpleShapeChecker();

foreach (var blob in blobs) {

     List<IntPoint> edgePoints = blobCounter.GetBlobsEdgePoints(blob);
     if (shapeChecker.IsQuadrilateral(edgePoints, out cornerPoints)) {
          if (shapeChecker.CheckPolygonSubType(cornerPoints) == PolygonSub-
          Type.Rectangle || shapeChecker.CheckPolygonSubType(cornerPoints) ==
          PolygonSubType.Square){

               blobIDs.Add(blob.ID);
               int blobSize = blob.Area;

               if (size < blobSize) {
                    size = blobSize;
                    biggestBlobID = blob.ID;
               }
          }
     }
}
```

Esimerkkikoodi 4. BlobCounterin etsimien blobien määrittäminen, SimpleShapeChecker-luokan käyttö ja suurimman blobin löytäminen.

BlobCounter-metodia käyttäessä löytyy useita ylimääräisiä kappaleita, jotka täytyy rajata pois lopputuloksesta. Ongelmatapauksessa ohjelma ei löydä ollenkaan kriteerien muotoisia kappaleita, jolloin se alkaa täyttämään ääriviivoja paksummiksi. Jokaisen ääriviivaa vastaavan pikselin kohdalla tarkistetaan, onko sillä vähintään kaksi samanväristä pikseliä vieressä. Jos vieressä olevia pikseleitä puuttuu, värjää ohjelma joko yhden ylemmän tai yhden oikealla olevan pikselin reunavärillä. Valittava pikseli määräytyy sen perusteella, missä suunnalla tarkasteltavaa pikseliä sijaitsee samanvärinen pikseli.

Kun kuvasta löytyneet ylimääräiset kappaleet on poistettu, kuvien suuntautuminen selvitetty ja kuvat käännetty pystysuuntaan (kts. seuraava kappale), kutsuu ohjelma AForgen suodatinta Crop. Crop-suodatin leikkaa suorakulmion muotoisen alueen alkuperäisestä kuvasta annettujen reunapisteiden avulla. Crop-metodille annetaan parametriksi suorakulmio, jolla on arvoina kuvan aloituspiste (x- ja y-koordinaattit) sekä leveys ja korkeus. (32.)

```
Crop(Rectangle(10,10,100,10));
```

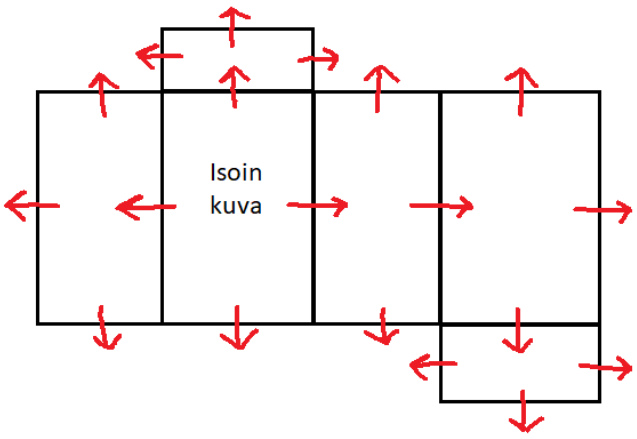
Esimerkkikoodi 5. Esimerkki Crop-metodin käytöstä.

Kun crop-metodia on kutsuttu, tallennetaan sivukuva Azuren kuvasäiliöön ja sivun tiedot tietokantaan. Tallennettavia tietoja ovat kuvan korkeus ja leveys, alkuperäinen ja nykyinen kuvan suuntautuminen, kuvan nimi, kuvan säiliöosoite sekä tieto siitä, mitä pakkauksen sivua kuva esittää.

5.5.5 Sivukuvan sijainti pakkauksessa

Sivukuvan sijainti pakkauksessa selvitetään BlobCounterin tekemän listan ja tallennettujen kulmapisteiden avulla. Lääkepakkauksen etusivuksi valitaan BlobCounterin tekemästä listasta suurin kuva. Suurin kuva valitaan etusivuksi, koska sen on todettu olevan enemmistö tapauksissa pakkauksen etusivu. Kuvan ympäriltä lähdetään katsomaan kulmapisteiden avulla, onko sillä vierustovereita. Vierustovereiksi lasketaan kuvat joiden kulmapisteet ovat toleranssin (valittu alue on 15 pikseliä) päässä toisistaan x- tai y-akseliin nähden.

Oletettavasti suurimmalla kuvalla on vähintään yksi vierustoveri. Vierustoverit etsitään katsomalla, onko isoimmalla kuvalla ylhäällä, alhaalla tai sivuilla siihen liittyviä kuvia. Jos kuvalta löytyy vierustovereita, tehdään sama tarkastelu myös niille ja niitä seuraaville uusille vierustovereille. Tarkastelu jatkuu, kunnes vierustovereita ei enää löydy.



Kuva 7. Isoimman kuvan vierustovereiden etsintää demonstroitu punaisilla nuolilla.

Prosessin aikana kuitenkin saattaa tallentua ylimääräisiä sivukuvia, jos lääkepakkaukselle on merkitty enemmän kuin kuusi sivua levityskuvaan. Ylimääräiset sivut ovat yleensä oikeiden sivujen alle taittuvia sivuja, joita ei oteta huomioon. Sovelluksen käyttäjä pystyy käyttöliittymän kautta poistamaan ylimääräiset sivut ja muokkaamaan ohjelman päättelemiä sivunimikeitä, jotka saattavat olla väärin ohjelman valitseman aloituskuvan (isoin kuva) takia.

5.5.6 Sivukuvien suuntautumisen selvittäminen

Levityskuvasta leikattavat sivukuvat saattavat olla väärinpäin tai sivulle kääntyneitä. Sivukuvat tahdotaan kuitenkin tallentaa kuvasäiliöön niin, että kaikki kuvat ovat oikeinpäin ja kuvassa olevat tekstit voidaan lukea. Apuna kuvien suuntautumisen selvittämiseksi käytetään .NET-versiolle tehtyä käärettä tesseract-ocr 3.04 tekstintunnistus-kirjastolle, joka mahdollistaa kirjaston käytön C#-kielessä (25).

Tesseract OCR lukee tessdata-kansioon tallennettujen kielitiedostojen avulla sille annettua kuvaa ja palauttaa kuvassa esiintyvän tekstin perusteella kuvan suuntautumisen. Tesseract palauttaa kuvan suuntautumisen arvoilla PageUp, PageDown, PageLeft ja PageRight (33). Jotta Tesseract pystyy lukemaan kuvasta sen suuntautumisen, täytyy Bitmap-muotoinen kuva lukea Tesseractin mukana tulevan Leptonica-kirjaston Pix-rakenteeseen. Tämän jälkeen Tesseract-moottori prosessoi kuvan ja palauttaa kuvan suuntautumisen AnalyseLayout-metodin avulla.

```
using (TesseractEngine engine = new TesseractEngine(path, "fin", Engine-
Mode.Default)) {

      //Convert Bitmap image to Pix
      Pix img = PixConverter.ToPix(bmp);
      engine.DefaultPageSegMode = PageSegMode.AutoOsd;

      //Preprocess image
      Page page = engine.Process(img);

      //Get Page Orientation
      Orientation orientation = page.AnalyseLayout().GetProperties().Orienta-
      tion;
}
```

Esimerkkikoodi 6. Tesseract-moottorin määrittely ja esimerkki kuvan suuntautumisen saamisesta. Moottorin määrittelyssä käytetty path-parametri kertoo moottorille, mistä kielitiedostot löytyvät.

Tesseractin palauttaman arvon perusteella kuvaa käännetään RotateFlip-metodilla. Metodille kerrotaan, kuinka monta astetta kuvaa kierretään ja käännetäänkö kuva esimerkiksi peilikuvaksi. Jos Tesseractin ilmoittama kuvasuunta on oikealle kääntynyt, kierretään kuvaa 270 astetta myötäpäivään, ja jos kuvasuunta on vasemmalle, käännetään kuvaa 90 astetta.

Jos kuvassa on liian vähän tekstiä, ei Tesseract pysty päättelemään kuvan suuntautumista, vaan se jättää kuvan käsittelemättä. Tällöin kuvalle asetetaan kuvan suunnaksi arvo ”undefined”.

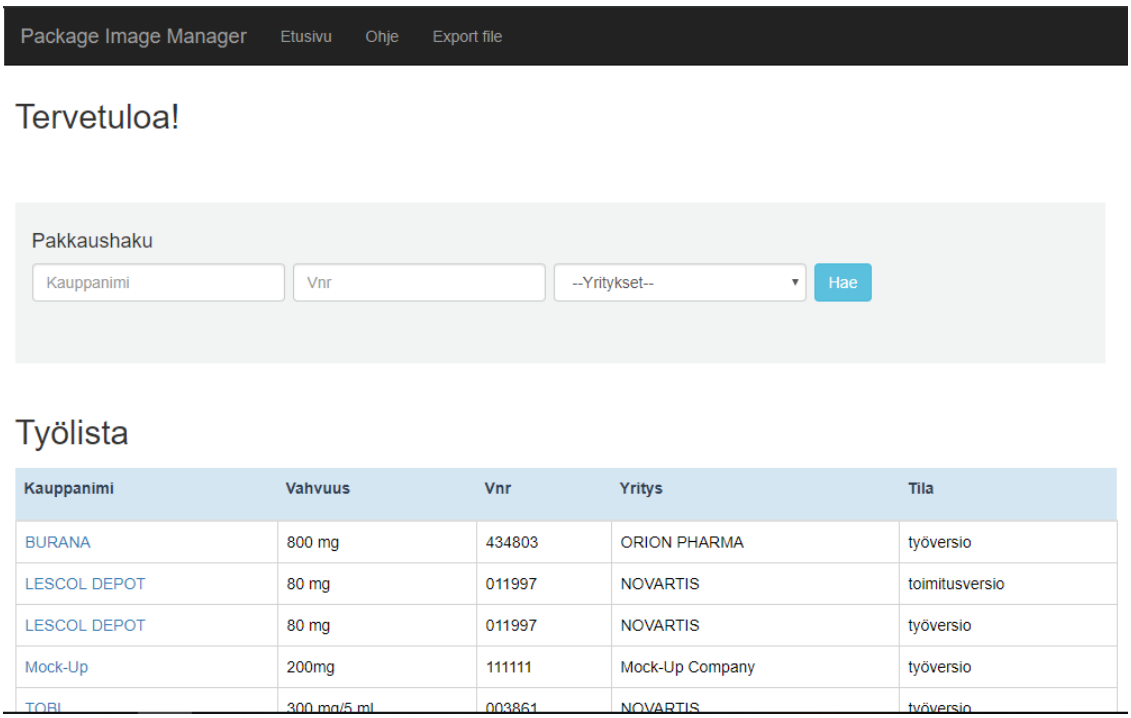
5.6 Käyttöliittymä

Käyttöliittymä on toteutettu sovellukseen .NET Framework v 4.6.1 -kehyksen avulla. Käyttöliittymä oli tarkoitus tehdä käyttäen .NET Framework Core 2.0 Jkehystä, mutta itse sovelluksessa käytetyt kirjastot (muun muassa Tesseract OCR) eivät olleet yhteensopivia kehyksen kanssa.

Käyttöliittymä on websovellus, joka mukailee MVC-mallia (Model-View-Controller), eli se koostuu näkymistä sekä kontrolleri- ja malliluokista. MVC-mallissa malliluokka edustaa kohdetta, jonka avulla siirretään tietoa näkymälle ja päivitetään kontrolleria. Kontrolleri toimii mallin ja näkymän välillä ohjaimena, ja pitää näkymän ja logiikan erillään. Sen tehtävänä on myös päivittää näkymää mallin tietojen muuttuessa.

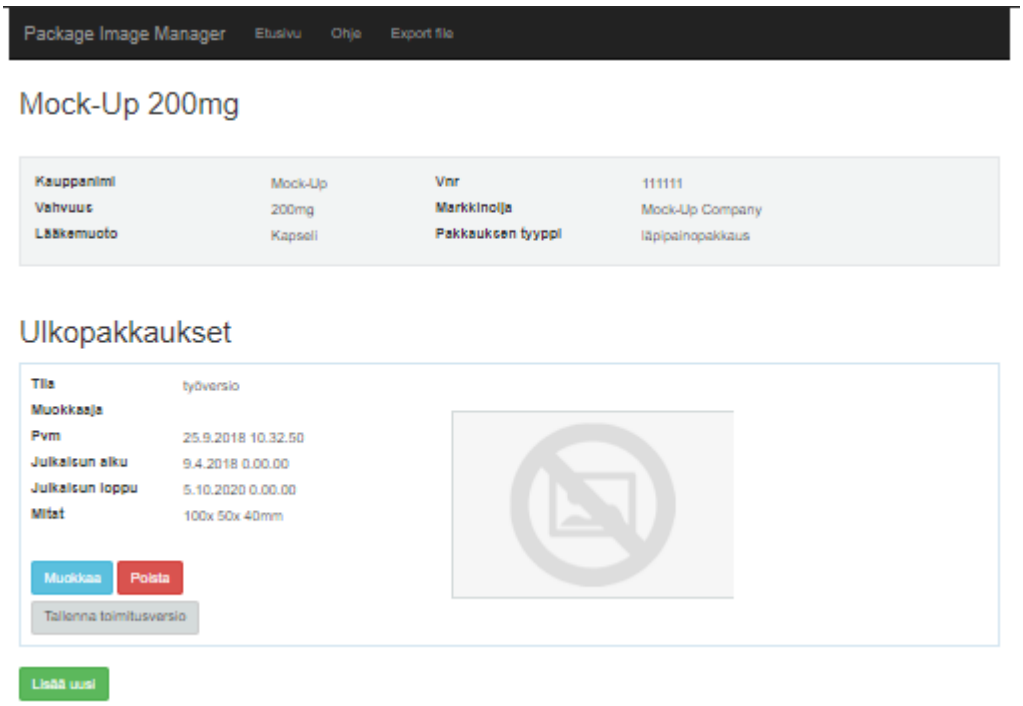
Käyttöliittymästä käyttäjä pystyy lataamaan ylhäältä löytyvän navigointipalkin kautta xml-muotoisen tiedoston, joka sisältää julkaisussa olevien ulkopakkausten tiedot. Ladattava tiedosto sisältää muun muassa kaikkien sivukuvien latausosoitteet, kuvien koon sekä ulkopakkauksen mitat ja julkaisuajat. Xml-tiedoston toteutus on tehty käyttämällä XmlWriter-kirjastoa, joka löytyy suoraan Visual Studion alkuperäiskirjastoista.

Käyttöliittymän aloitussivulla on haku, josta voi etsiä lääkepakkauksia kauppanimen, vnr-numeron ja yrityksen mukaan. Kaikkia hakutermejä voidaan käyttää samanaikaisesti oikean hakutuloksen saamiseksi. Aloitussivulla on haun lisäksi myös työlista, jossa näytetään kaikki pakkaukset, joilla on käsittelemättömiä ulkopakkauksia. Käsittelemättömiä ulkopakkauksia ovat ulkopakkaukset, joiden status on työversio tai toimitusversio.



Kuva 8. Käyttöliitymän aloitussivu.

Kun pakkaus on valittu joko työlistasta tai hakutuloksista, näytetään sivulla pakkauksesta löytyvät erilaiset ulkopakkaukset.

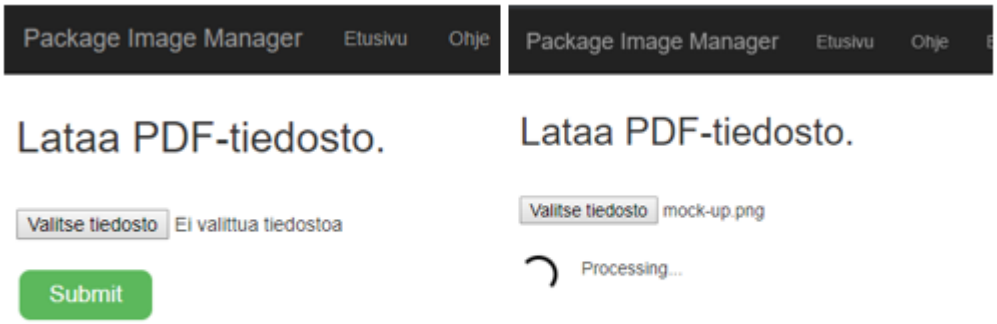


Kuva 9. Lista lääkevalmisteen ulkopakkauksista.

Ulkopakkausten listasta käyttäjä pystyy nopeasti katsomaan pakkaukseen liittyvät sivukuvat klikkaamalla ulkopakkauksen kuvaa, joka näytetään ulkopakkaustietojen oikealla puolella. Ylhäällä olevassa kuvassa (kts. kuva 9) ulkopakkauksen 3D-kuvaa ei ole vielä saatavilla, vaan käyttäjälle näytetään kuva, joka ilmoittaa, ettei pakkauskuvaa ole saatavilla. Kuvaa klikatessa, käyttäjälle näytetään pop-up-ikkuna, jossa näytetään kaikki ulkopakkauksen sivukuvat. Tästä käyttäjä pystyy tarkistamaan, että kaikki kuvat ovat oikeinpäin ja niitä on oikea määrä.

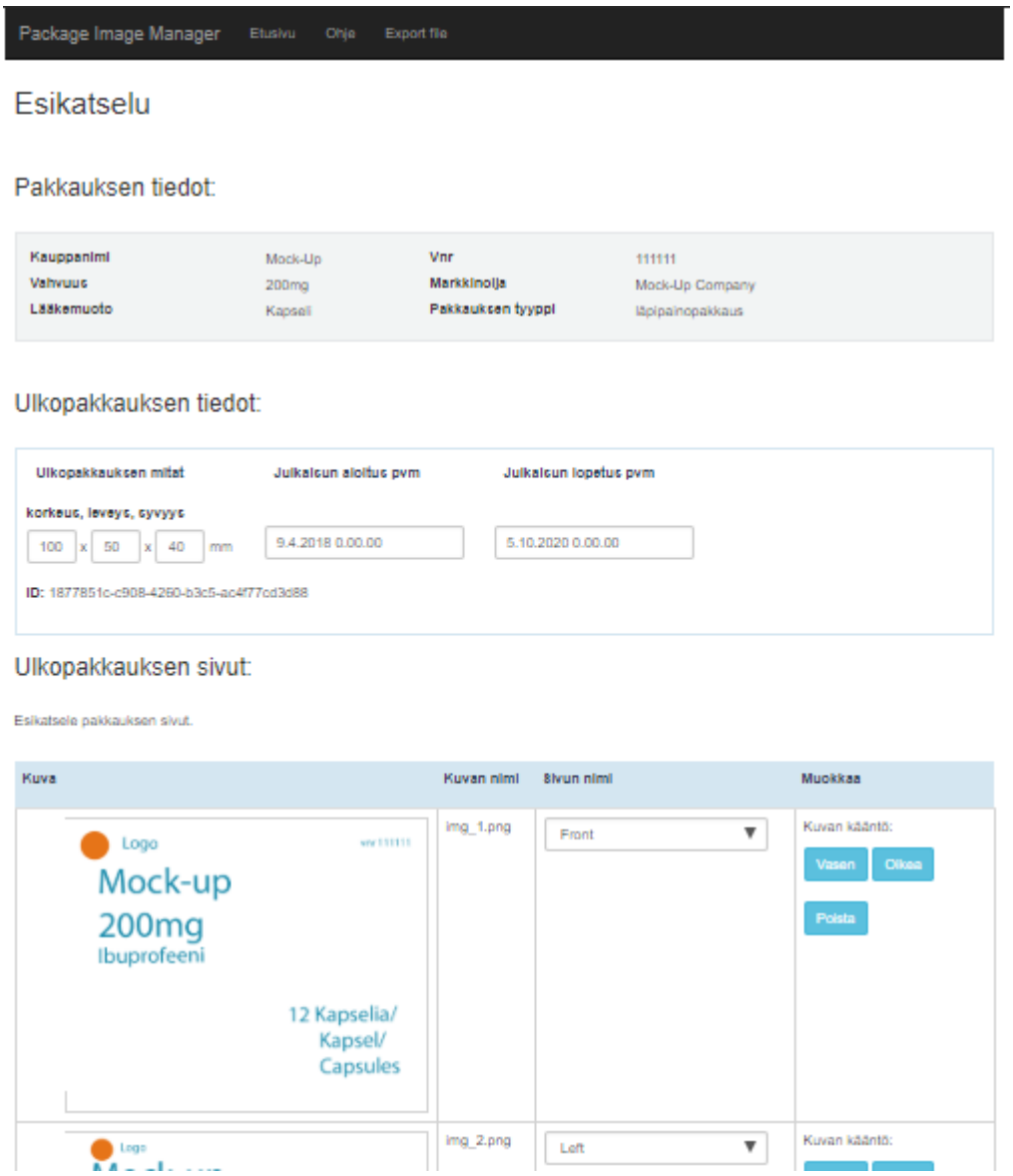
Pop-up-ikkuna on toteutettu osittaisella näkymällä, jota kutsutaan linkillä, johon on upotettu kuva. Linkki kutsuu JavaScript-metodia, joka populoi näkymässä olevan osittaisen näkymän sisällöllä. JavaScript-metodille annetaan parametriksi ulkopakkauksen id, jonka perusteella se hakee ulkopakkauksen sivut kontrollerissa ja lisää ne osittaisen näkymän sisällöksi.

Olemassa olevia pakkauksia voi muokata tai pakkaukselle voi lisätä uusia ulkopakkauksia. Uuden kuvan lisäysprosessissa annetaan palvelulle ensin tiedosto, joka on joko png- tai pdf-muotoinen. Painettaessa ”Submit”-painiketta itse ohjelma alkaa pyöriä ja erottelee tallennetusta tiedostosta pakkauksen sivut.



Kuva 10. Uuden ulkopakkauksen lisäämisen ja prosessoinnin näkymät.

Ohjelman valmistuessa käyttäjä siirretään muokkausnäkymään, jossa käyttäjälle näytetään pakkauksen sivukuvat ja niihin liittyvät tiedot. Muokkausnäkymän kautta käyttäjä voi muuttaa tietoja ja poistaa ylimääräisiä sivukuvia. Muokkausnäkymän kautta ulkopakkaukselle lisätään puutuvat tiedot; pakkauksen mitat sekä julkaisupäivämäärä ja päivämäärä, jolloin pakkaus on poistettu markkinoilta.



Kuva 11. Ulkopakkauksen sivukuvien esikatselu ja muokkausnäkymä.

5.7 Testiympäristö

Projektille tehtiin pilveen testiympäristö, joka pyörii Azuren tuottamassa App Service -pilvipalvelussa. App Service on palvelu, joka ylläpitää muun muassa verkkosovelluksia ja REST-sovellusliittymiä. Palvelu tarjoaa useille eri kielille pilvipalvelussa alustan, jossa ne toimivat ilman infrastruktuurin hallintaa. Palvelu tarjoaa automaattisen skaalauksen ja korkean käytettävyyden, tukee sekä Windowsia että Linuxia ja mahdollistaa automatisoidut asennukset esimerkiksi GitHubista. (34.) Testiympäristön tarkoituksena oli mahdol-

listaa koko Lääketietokeskuksen henkilöstön pääsy sovellukseen. Testiympäristön toteutettiin kirjautuminen Azuren Active Directoryn kautta, joka vaatii Microsoftin Outlook -sähköpostiosoitteen, joka on kirjattu Lääketietokeskuksen nimiin.

5.8 Projektin aikana kohdatut ongelmat

Projektia tehdessä vastaan tuli ongelmia pdf-kirjaston, materiaalien ja testiympäristön pystyttämisen kanssa. Kaikkiin ongelmiin löydettiin ratkaisu.

5.8.1 Pdf-kirjasto

Projektiin valittavan sopivan pdf-kirjaston löytäminen oli hankalaa. Kokeilluista yhdeksästä eri kirjastosta vain yksi osoittautui käyttökelpoiseksi projektiin. Suurin osa kirjastoista ei pystynyt rasterisoimaan eli muuntamaan pdf-tiedostoa kuvaksi, ja ne kirjastot jotka pystyivät, eivät osanneet siirtää tiedoston värejä oikein kuvaan. Sinisestä tuli muun muassa oranssia ja punaista muunnoksen aikana. ABCPdf-kirjaston lisäksi kokeiltiin seuraavia kirjastoja:

- Acrobat Reader Plug-in
- PDFSharp
- Free Spire.PDF
- ITextSharp
- PDFBox.NET
- PDFClown
- IronPDF
- Byte Scout PDF Renderer SDK.

Acrobat Reader Plug-in-liitännäinen muodosti kaikista kymmenestä kirjastosta parhaimman tuloksen. Sen kuvan laatu oli todella hyvä ja värit pysyivät samoina, eikä värialueiden pikselien välillä ollut sävyjen muuntumista toiseksi. Acrobat Reader -liitännäisen ongelmaksi muodostui se, että se vaati lokaalin asennuksen Acrobat Readerista, mitä ei voitu siirtää pilvessä toimivaan Azuren ympäristöön. Toisena ongelmana olivat liian suuret ja tarkat pdf-tiedostot, joita liitännäinen ei pystynyt käsittelemään. Näiden suurien tiedostojen käsitteleminen ei onnistunut edes itse Acrobat Reader -ohjelman kautta. PDFSharp-kirjasto aiheutti samat ongelmat, sillä se käytti kuvamuunnokseen Acrobat Reader -ohjelmaa myös.

Free Spire.PDF -kirjaston tekemissä kuvamuunnoksissa kuvien tekstit olivat epätarkkoja ja kuvien värit olivat täysin erilaiset mitä alkuperäisessä kuvassa. Sinisestä tuli oranssia ja pinkkiä riippuen värin sävystä. Kirjasto lisäsi syntyviin kuviin vaalean harmaita viivoja, jotka eivät kuuluneet kuvaan. Kirjaston ainoana hyvänä puolena olisi ollut se, että se pystyi käsittelemään suuriakin tiedostoja vaivattomasti. Samanlaisia väriongelmia aiheuttivat IronPDF- ja Byte Scout PDF Renderer SDK -kirjastot. Näissä kahdessa kirjastossa kuvanlaatu oli muuten hyvä ja tekstit luettavia.

PDFBox.NET on .NET-ohjelmointiin tehty kääre saman nimiselle Java-kirjastolle. Kääre pystyi lukemaan pdf-tiedostosta tekstit useiden virheiden kanssa, muttei pystynyt tekemään kuvamuunnosta .NET-versiossa. PDFClown-kirjastoa ei saatu toimimaan kunnolla, mikä johtui luultavasti sitä, että se oli vielä kehitteillä beta-vaiheessa. Kirjaston tarjoama Renderer-luokka ei pystynyt renderöimään pdf-tiedostoa, vaikka dokumentaation mukaan sen olisi pystyttävä toimenpiteeseen.

5.8.2 Vialliset materiaalit

Osa materiaaleista on koottu kerros kerrokselta, eli jollain kerroksella on ulkopakkauksen ääriviivat ja jollain siihen liitetyt kuvat. Näissä tapauksissa ääriviiva oli jäänyt jonkin toisen kerroksen alle, eikä pdf-kirjasto osaa lukea kerroksen tasoa oikein rasterisaation yhteydessä.

Ongelmaa aiheuttivat myös pakkaukset, joiden ääriviiva sisälsi katkoviivaa jatkuvan viivan sijaan. AForge-kirjasto ei tällöin mieltänyt syntyvää kuviota suorakulmioksi ja jätti nämä palaset huomioimatta etsiessään oikeanmuotoisia kappaleita.

Ratkaisuksi näihin ongelmiin muodostui se, että materiaalit tarkistetaan ja muokataan tarvittaessa, jos ne eivät mene ohjelman läpi. Tällöin tarpeen vaatiessa käyttäjä tarkistaa, että ulkopakkauksen ääriviivat ovat tiedoston ylimmällä tasolla, sekä muokkaa tiedoston katkoviivat yhtenäisiksi.

5.8.3 Testiympäristön toimintaan saaminen

Projektissa syntyneelle ohjelmalle rakennettiin pilvessä pyörivä testiympäristö, jonne lokaalisti toimiva projekti siirrettiin. Testiympäristön oli tarkoitus toimia täysin samalla tavalla kuin lokaalisti pyörivä ohjelma. Kuitenkaan siirto ei onnistunut täysin, sillä ohjelman

tärkein ominaisuus, eli sivukuvien irrotus ei toiminut. Ensimmäiseksi havaittiin, että pilvessä pyörivä sovellus ei pystynyt käyttämään pdf-kirjaston lisenssiä, josta saatiin varoitus Google Chromen kehittäjätyökalujen avulla. Ongelma korjattiin lisäämällä ohjelmakoodiin asetus, joka antaa koodille jokaisella ajokerralla käyttöoikeuden käyttää lisenssiä. Asetus on ABCpdf-kirjaston oma konfigurointi asetus, jota kutsutaan ennen itse kirjaston kutsumista koodissa.

```
XSettings.InstallLicense(”lisenssikoodi”);
```

Esimerkkikoodi 7. Pdf-kirjaston lisenssin käyttöoikeuden myöntäminen ohjelmakoodin suorituksen aikana.

Toiseksi ongelmaksi muodostui sovelluksen ajon kesto. Lokaalisti kirjastoprojektin runmetodin ajoaika on noin 60 sekuntia. Kuitenkin tästä saatiin kehittäjäntyökalujen avulla virhe serverin aiheuttamasta aikakatkaisusta, joka tapahtui 60 sekunnin jälkeen. Ajon aikaa saatiin lokaalisti pienennettyä 30 sekuntiin, mutta ongelma pysyi samana. Useiden testien ja hakujen jälkeen syyksi selvisi, että syy johtuu Azuren palvelusta. Ongelma kierrettiin toisella AJAX-kutsulla, joka kyselee ensimmäisen AJAX-kutsun suoritettavalta metodilta, onko kutsu valmistunut.

AJAX (Asynchronous JavaScript And XML) on kokoelma websovelluskehityksen tekniikoita, jonka avulla websovelluksista voi tehdä vuorovaikutteisempia. Ajaxissa selaimessa toimiva ohjelma vaihtaa taustalla pieniä määriä dataa palvelimen kanssa niin, ettei koko verkkosivua tarvitse ladata uudelleen joka kerta, kun jokin pieni asia muuttuu sivulla. (35.) Tätä kutsutaan termillä SPA (Single-Page Application). SPA eli yksisivuinen sovellus on verkkosivusto, joka palauttaa sisältönsä vastauksena navigointitoimiin (esimerkiksi linkin napsauttamiseen) pyytämättä palvelimelle uutta HTML-koodia. Tämä tarkoittaa sitä, että kaikki sivun sisältämät tiedot ladataan kerralla selaimeen, mutta kaikkea ladattua tietoa ei näytetä heti. (36.)

6 Jatkokehitys

Sovelluksen kehittäminen jatkuu yrityksessä ja sen on tarkoitus tulla tuotantokäyttöön sen valmistuttua täysin. Sovellukseen on tarkoitus lisätä ulkopakkausten 3D-kuvat, jotka muodostetaan tämän sovelluksen avulla syntyvistä ulkopakkausten sivukuvista. Sovellukseen on suunnitteilla myös näkymä, jossa ulkopakkausta pystyy pyörittelemään 3D-mallima. Sovellus on suunniteltu aluksi Lääketietokeskuksen sisäiseen käyttöön, mutta

käyttöä on tarkoitus laajentaa myös Lääketietokeskuksen asiakkaille. Tätä varten sovellukseen pitää vielä tehdä joitakin muutoksia.

Ohjelman jatkokehityksen myötä tämä projekti auttaa yritystä jakamaan sen asiakkaille pakkauksien 3D-malleja, sekä niihin liittyviä tietoja, jotka ovat esimerkiksi varastonhallinnassa erittäin tärkeitä. Jatkokehityksen lisätavoitteina on säästää työaikaa (ja vähentää muuten manuaalisessa työssä mahdollisesti tapahtuvia virheitä) aineiston keräämisessä ja ylläpidossa sekä tarjota uusia mahdollisuuksia Lääketietokeskukselle ja sen asiakkaille.

7 Yhteenveto

Insinöörityön tuloksena syntyi toimiva sovellus, joka erottelee sille annetuista pakkausmateriaaleista sen esittämän ulkopakkauksen sivukuvat. Ohjelman ympärille on rakennettu selkeä käyttöliittymä, jota kautta käyttäjän on helppo työstää ja hallinnoida järjestelmässä olevia ulkopakkaustietoja. Ohjelmassa perustietoina olevat lääkevalmisteiden tiedot pystytään päivittämään erillisen Visual Studio projektin kautta aina uuden lääkevalmisteen tullessa markkinoille tai jo olemassa olevan valmisteen tietojen muuttuessa.

Projektin tavoitteina oli toteuttaa sovellus, joka automatisoi lääkevalmisteiden ulkopakkausmateriaalien käsittelyprosessia ja minimoi henkilöstön käyttämää aikaa materiaalien käsittelyyn. Kolmantena tavoitteena sovelluksen tarkoituksena oli helpottaa ulkopakkausten jakeluprosessia yrityksen asiakkaille tulevaisuudessa. Projektille annetut tavoitteet toteutuivat hyvin. Sovelluksen ansiosta henkilöstön ei tarvitse käyttää aikaa siihen, että materiaaleista poimitaan käyttökelpoiset osiot manuaalisesti ja tallennetaan kaikki niihin liittyvä informaatio tietokantaan ja pilveen. Ulkopakkausmateriaalien käsittelyprosessi on lähes automaattinen, jonka ansiosta henkilöstön työskentelyn tehostamisen lisäksi voidaan välttyä osalta manuaalisessa työssä syntyviltä virheiltä. Rajapinnasta haettavat perustiedot (vnr-numero, valmisteen kauppanimi ym.) auttavat muun muassa välttämään kopiointivirheitä, sillä materiaaleista ei tarvitse rajapinnan ansiosta syöttää tietoja ohjelmaan.

Insinöörityön ansiosta opin paljon uutta. Työn ohella opin uuden ohjelmointikielen pinnallisesti ja insinöörityön tarjoamien oppien myötä pystyn jatkossa kehittämään sitä jo opitun perusteella. Työn ansiosta pääsin oppimaan myös uudenlaisia tekniikoita ja käyttämään minulle vieraita kirjastoja, kuten Tesseract OCR - ja Entity Framework -kirjastoja.

Lähteet

1. Szeliski, Richard. 2010. Computer Vision: Algorithms and Applications. Verkkoaineisto. <http://szeliski.org/Book/drafts/SzeliskiBook_20100903_draft.pdf> . 3.9.2010. Luettu 06.09.2018.

2. Peters, James F. *Foundations of Computer Vision.* s.l. : Springer International Publishing 2017. s. 412. ISBN 3-319-52481-X.

3. Pitas, Ioannis. 2000. *Digital Image Processing Algorithms and Applications.* ISBN 0-471-37739-2.

4. AForge.NET Framework. Verkkoaineisto. <http://www.aforgenet.com/framework/>. Luettu 18.09.2018.

5. AForge.NET Framework Features. Verkkoaineisto. <http://www.aforgenet.com/framework/features/>. Luettu 18.09.2018.

6. AForge.NET. Blobs Processing. Verkkoaineisto. <http://www.aforgenet.com/framework/features/blobs_processing.html>. Luettu 01.10.2018.

7. Kirillov, Andrew. 2012. *BlobCounter.cs.* Verkkoaineisto. <https://github.com/cureos/aforge/blob/master/Sources/Imaging/BlobCounter.cs>. 18.02.2012. Luettu 02.10.2018.

8. AForge.NET. 2009. Blob Operation. Verkkoaineisto. <http://www.aforgenet.com/forum/viewtopic.php?f=4&t=1292>. 23.01.2009. Luettu 02.10.2018.

9. Kemp, Karen K. 2007. Encyclopedia of Geographic Information Science. Verkkoaineisto.<https://books.google.fi/books?id=clF2AwAAQBAJ&pg=PT39&dq=binary+large+object&hl=fi&sa=X&ved=0ahUKEwjonP3i7bDdAhXEliwKHRZKAUMQ6AEIZTAI#v=onepage&q=binary%20large%20object&f=false>. Luettu 10.09.2018.

10. Kesheng, Wu;Otoo, Ekow ja Suzuki, Kenji . Optimizing Two-Pass Connected-Component Labeling Algorithms. Verkkoaineisto. <https://sdm.lbl.gov/~kewu/ps/paa-final.pdf>. Luettu 02.10.2018.

11. Schwenk, Kurt ja Huber, Felix. Connected Component Labeling Algorithm for very complex and high resolution images on an FPGA platform. Verkkoaineisto. <https://elib.dlr.de/100363/1/published_version.pdf>. Luettu 02.10.2018.

12. Chaudhuri, Arindam;ym. 2017. *Optical Character Recognition Systems for Different Languages with Soft Computing.* E-Kirja. s.l. : Springer International Publishing, 2017. ISBN 978-3-319-50252-6 (eBook).

13. Mohammad, Faisal ;ym. 2014. *Optical Character Recognition Implementation Using Pattern Matching.* Verkkoaineisto*.* *<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.661.1089&rep=rep1&type=pdf>. Luettu 03.09.2018.*

14. Rouse, Margaret. 2005. *ASCII (American Standard Code for Information Interchange).*Verkkoaineisto.< https://whatis.techtarget.com/definition/ASCII-American-Standard-Code-for-Information-Interchange>. Luettu 03.09.2018.

15. Tesseract OCR. Verkkoaineisto. <https://github.com/tesseract-ocr/tesseract>. Luettu 04.09.2018.

16. Improving the quality of the output. 2018. Verkkoaineisto. <https://github.com/tesseract-ocr/tesseract/wiki/ImproveQuality>. Luettu 04.09.2018.

17. ApiExample. 2018. Verkkoaineisto. <https://github.com/tesseract-ocr/tesseract/wiki/APIExample>. Luettu 04.09.2018.

18. Smith, Ray . 2007. *An Overview of the Tesseract OCR Engine.* Google Inc. Verkkoaineisto.<https://storage.googleapis.com/pub-tools-public-publication-data/pdf/33418.pdf>. Luettu 04.09.2018.

19. Nordic Article Number (Vnr). Verkkoaineisto. Lääketietokeskus Oy. <http://wiki.vnr.fi/?page_id=36>. Luettu 08.08.2018.

20. Kotkanen, Henri. 2016. Mikä on (avoin) data? Helsingin Kaupunki. Verkkoaineisto. <https://www.hel.fi/hel2/tietokeskus/data/dokumentit/koulutusmateriaali/Hyodyntamiskoulutus_Henri_Kotkanen.pdf> 24.04.2016. Luettu 20.09.2018.

21. Vega, Diego;ym. 2016. Code First Migrations. Verkkoaineisto. <https://docs.microsoft.com/en-us/ef/ef6/modeling/code-first/migrations/>. 23.10.2016. Luettu 03.10.2018.

22. Toub, Stephe. 2017. Performance Improvements in .NET Core. Verkkoaineisto. <https://blogs.msdn.microsoft.com/dotnet/2017/06/07/performance-improvements-in-net-core/>. 07.06.2017. Luettu 24.09.2018.

23. Vega, Diego;Wenzel, Maira ja Miller, Rowan. 2018. Entity Framework 6. *Microsoft Docs.* Verkkoaineisto. <https://docs.microsoft.com/en-us/ef/ef6/>. 27.08.2018. Luettu 28.08.2018.

24. An Introduction To Entity Framework Core. 2017. Verkkoaineisto. <https://www.learnentityframeworkcore.com/>. 24.02.2017. Luettu 28.08.2018.

25. Weld, Charles. Tesseract. Verkkoaineisto. <https://github.com/charlesw/tesseract>. Luettu 20.08.2018.

26. Tesseract. 2017. Verkkoaineisto. <https://www.nuget.org/packages/Tesseract/3.2.0-alpha4>. 23.08.2017. Luettu 04.09.2018.

27. Ibrahim, Mohammed. 2016. OCR using Tesseract in C#. Verkkoaineisto. <https://www.c-sharpcorner.com/article/ocr-using-tesseract-in-C-Sharp/>. 07.03.2016. Luettu 24.09.2018.

28. Rasterization and Vectorization: How to Convert Data Formats. 2018. Verkkoaineisto. <https://gisgeography.com/rasterization-vectorization/>. 23.02.2018. Luettu 09.10.2018.

29. Dictionary<TKey,TValue> Class. Verkkoaineisto. <https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.dictionary-2?view=netframework-4.7.2>. Luettu 10.10.2018.

30. Color.ToArgb Method. Verkkoaineisto. <https://docs.microsoft.com/enus/dotnet/api/system.drawing.color.toargb?redirectedfrom=MSDN&view=netframework-4.7.2#System_Drawing_Color_ToArgb>. Luettu 10.10.2018.

31. AForge.NET. BlobCounter Class. Verkkoaineisto. <http://www.aforgenet.com/framework/docs/html/d7d5c028-7a23-e27d-ffd0-5df57cbd31a6.htm>. Luettu 06.09.2018.

32. Crop Class. Verkkoaineisto. <http://www.aforgenet.com/framework/docs/html/197f820b-f5d0-57cb-a509-5dbcacdda446.htm>. Luettu 06.09.2018.

33. Orientation Enumeration. Verkkoaineisto. <https://tesseract.patagames.com/help/html/T_Patagames_Ocr_Enums_Orientation.htm>. Luettu 06.09.2018.

34. Web Apps overview. 2017. Verkkoaineisto. <https://docs.microsoft.com/en-us/azure/app-service/app-service-web-overview>. 04.01.2017. Luettu 07.11.2018.

35. Morris, Scott. 2018. AJAX—What It Is, How It Works, and What It's Used For. Verkkoaineisto. <https://skillcrush.com/2018/04/26/what-is-ajax/>. 25.04.2018. Luettu 12.10.2018.

36. Sherman, Paul. 2018. How Single-Page Applications Work. Verkkoaineisto. <https://medium.com/@pshrmn/demystifying-single-page-applications-3068d0555d46>. 11.04.2018. Luettu 07.11.2018.