# Abstract

The purpose of this thesis is to create a mobile application for expense tracking, with the main focus on functionality allowing to take pictures of receipts issued by Estonian enterprises, extract basic expense information from the captured receipt images and store it in authenticated user’s expense list.

The main problems covered in this work are finding the best architectural and design solutions for the application from the perspective of performance, usability, security and further development as well as researching and implementing techniques to handle expenses recognition from receipts in an efficient way.

As a result of the thesis, a working implementation of expense tracking mobile application for Android appears. After functionality of expense information extraction from receipt image passes the testing phase, conclusion regarding its reliability is made. Moreover, proposals for further improvements of the application’s functionality are also presented.

# Annotatsioon

Käesoleva töö eesmärgiks on luua mobiilirakendus kasuataja kulude pidamiseks ja dokumenteerimiseks, peamiselt fokuseerides funkstionaalsusele, mis peab lubama teha pilte Eesti ettevõtete poolt väljastatud ostutšekkidest, lugeda väja kulu baasinformatiooni ostkutšekki pildi pealt ning salvestada saadud informatsiooni autentitud kasutaja kulude listi.

Töö põhilisteks probleemideks on sobivate rakenduse arhitektuuri -ja disainilahenduste väljatöötamine rakenduse jõudluse, kasutatavuse, turvalisuse ja edasiarenduse perspektiividest ning ostutšekki pildi pealt kulu info väljalugemise effektiivsete mehhanismide uurimine ning implementeerimine.

Töö tulemusena valmib mobiilirakendus kulude pidamiseks Android platvormile. Peale seda, kui rakenduse funktsionaalsus kulude väljalugemiseks ostutšekkide piltide pealt läbib testmisfaasi, tehakse järeldusi selle toimimise ja kindluse kohta. Lisaks tuuakse välja ka võimalikke edasiarenduvõimalusi ja parandusi valminud mobiilirakenuse jaoks.

# Glossary of terms and abbreviations

|  |  |
| --- | --- |
| **API** | Optical charachter recognition |
| **APK** | Android application package |
| **CPU** | Central processing unit |
| **CRUD** | Create, read, update and delete |
| **FIFO** | First In, First Out |
| **Google Developers Console** | Google environment for binding Android application with Google services |
| **HTML** | Hypertext Markup Language |
| **HTTP** | Hypertext Transfer Protocol |
| **HTTPS** | HTTP over SSL |
| **JDBC** | Java database connectivity |
| **JSON** | JavaScript Object Notation |
| **OCR** | Optical charachter recognition |
| **ORM** | Object-relational mapping |
| **REST** | Representational State Transfer |
| **Scraper** | A program assembling content from web pages |
| **SDK** | Software development kit |
| **SQL** | Structured Query Language |
| **SSL** | Secure Sockets Layer |
| **XML** | Extensible Markup Language |

# Introduction

Today there are relatively many mobile applications for expense tracking. It goes without saying, that one of the core features that all the expense tracking applications have is a possibility to manually submit basic expense information (eg. total sum, merchant/company etc.). But with the development of modern technologies, some expense tracking applications introduced an opportunity to take pictures of purchase receipts in order to automatically recognize and extract basic expense information from the receipt image and place the extracted information into user’s expense list inside the application. From the users’ perspective, such functionality may be quite useful, as it prevents them from manual expense submission in favor of automatic receipt processing. The most well-known examples of applications of such kind are Expensify[1] and Xpenditue[2]. But at the same time, these applications are proprietary and what is more, they are not free of charge when it comes to extensive usage of expenses extraction from receipt image. By the time the author started with this thesis, he has not found any noted open-source expense tracking software with the functionality of expense recognition from a receipt image. Because of this, the author decided to implement such mobile application, which would allow expense data recognition from images of receipts issued in Estonia, using open-source technology stack.

## Problem and background

The main problem the author focuses in this work is to find out the best designing solution for expense tracking mobile application, focusing on possibility of expense information recognition from Estonian receipts as well as usability and security aspects of the application.

The reason why this work may be considered useful for the broader audience is that as a result of it appered a mobile application, which can simplify users’ everyday expenses tracking and thus to help planning personal budget more sensibly. In perspective it can be used either as a standalone application, or be integrated with some mobile banking application.

The knowledge gained during development of this application and pointed out in this work may be useful from the perspective of a software engineer, as it covers the following topics:

* Comparison of potential solutions regarding the architecture and design of such kind of application.
* Methods of receipt digital image processing in order to improve recognition of the text of receipt
* Complexities and contsraint related to implementation of an application of such kind.

## Goals

The main goal of this work is to implement a mobile application with the following functionality:

1. Possibility to take picture of a receipt, recognize total sum of the purchase and the company the purchase was made from based on the receipt image, compose an expense entry from this information and place it into user’s expense list.
2. Possibility for the user to see his own expenses.
3. Possibility to insert expense entry manually, which can be useful in case recognition of expense from receipt has failed due to some reason.
4. Possibility to make pictures of receipts while device is not connected to the internet and make expense recognition from those pictures later.

## Methodology

To achieve the goal the author implemented the Android client-side application and the server side application written in Java programming language. Both client and server are designed in an object-oriented manner. The server side is put to communicate with the database (PostgreSQL) and an external web service of the Estonian e-business register. Both service and client are using third-party libraries.

## Thesis overview

The second part of the thesis covers the theoretical background of image preprocessing and OCR operations, which are used in the process of expense extraction from a receipt image.

The third part of the thesis covers analysis of possible architecture and design solutions of the application and explains why the chosen solutions are the most suitable in a context of the application.

The fourth part of the thesis introduces a brief overview of the chosen technologies. It is also explained, for which puproses each technology was chosen.

The fifth part of the thesis covers the topic of the application’s implementation phase. It is explained how the key components of the application were implemented with the use of selected technologies.

The sixth part of the thesis evaluates the core functionality of the implemened application, analyses its reliability as well as proposes possible further improvements, which should be considered in the future development.

# Theory

## Image preprocessing before optical character recognition

In the author’s mobile application, the picture of receipt used for further expense information extraction is taken using the mobile device’s camera. As not all the mobile devices have high resolution cameras, this means that the quality of the captured receipt image itself may not be good enough to perform successful optical character recognition. Besides that, the recognition stage is made even harder by the fact that cash registers use mainly either thermal printers with thermal paper or dot matrix printers for receipt printing. The problem of the receipts printed with the thermal printer is that they tend to fade with time, and the problem of the dot-matrix-printed receipts is that the characters are composed of small dots, which makes it hard for the OCR engine to determine them correctly. As a result, in order to enhance the quality of the image with the purpose of better OCR results, image preprocessing should be applied.

### Conversion to grayscale

The first measure to be taken is order to enhance image for further OCR is converting color image to grayscale image. Grayscale image is an image in which the value of each [pixel](http://en.wikipedia.org/wiki/Pixel) it carries only [intensity](http://en.wikipedia.org/wiki/Luminous_intensity) information. Images of this sort, also known as [black-and-white](http://en.wikipedia.org/wiki/Black-and-white), are composed of shades of [gray](http://en.wikipedia.org/wiki/Grey), varying from black at the weakest intensity to white at the strongest. (Stephen Johnson (2006). [*Stephen Johnson on Digital Photography*](http://books.google.com/books?id=0UVRXzF91gcC&pg=PA17&dq=grayscale+black-and-white-continuous-tone&ei=XlwqSdGVOILmkwTalPiIDw). O'Reilly. [ISBN](http://en.wikipedia.org/wiki/International_Standard_Book_Number) [0-596-52370-X](http://en.wikipedia.org/wiki/Special:BookSources/0-596-52370-X)) Conversion of the image to grayscale is important from the perspective of making further image processing stages faster.

([http://research.ijcaonline.org/volume55/number10/pxc3882784.pdf p. 52](http://research.ijcaonline.org/volume55/number10/pxc3882784.pdf%20p.%2052)).

### Denoising

The second stage of preprocessing is denoising. Noise is considered to be either error in the pixel value or an erroneous bit pattern with no significance in the output image, which is takes place during the acqusition process of the image. The noise may be amplified by the digital corrections of the camera or tools removing blur or increasing contrast of the images.

The most trivial denoising method is replacing the color of the pixel with an average of the colors of nearby pixels. But in practice it does not often lead to the desired result, as similar pixels are not always close to each other.  Thus it proves to be more reliable to scan a vast portion of the image in order to find all the pixels that really resemble the pixel to be denoised. Denoising is then done by calculating the average color of these most similar pixels. The similarity is evaluated by comparing a whole window around each pixel, and not just the color. The filter performing such kind of operation is called non-local means filter (Antoni Buades, Bartomeu Coll, and Jean-Michel Morel, Non-Local Means Denoising, [Image Processing On Line](http://dx.doi.org/10.5201/ipol), [1](http://dx.doi.org/10.5201/ipol.2011) (2011).<http://dx.doi.org/10.5201/ipol.2011.bcm_nlm>). This filter is applied by the author in his practical part of the work using the OpenCV library.

### Morphological closing

As it was mentioned above, the quality of receipt print is quite frequently of low level. It is particularly reflected in the fact that some of the printed symbols contain gaps, which isolate parts of a single symbol. As a result, the OCR engine tends to misinterpret separate parts of a single symbol as separate symbols. For the puprose of linking the parts of a symbol separated with the gaps, an operation of morphological closing(<http://docs.opencv.org/doc/tutorials/imgproc/opening_closing_hats/opening_closing_hats.html>) should be applied.

This operation is comprised of two basic morphological operations: dilation, followed by erosion.(http://docs.opencv.org/doc/tutorials/imgproc/erosion\_dilatation/erosion\_dilatation.html) As a result of applying morphological closing, parts of the symbol which were separate before tend to become connected as well as dots inside the symbols tend to get removed.

<http://opencv-python-tutroals.readthedocs.org/en/latest/py_tutorials/py_imgproc/py_morphological_ops/py_morphological_ops.html>

### Thresholding

The operation of thresholding is considered to be a simple method of image segmentation. It takes a grayscale image as an input and transforms it into a binary image, comprised of only two colors, according to the following logic: if the value of a pixel is below the determined threshold value, then the pixel is assigned the minimum value (e.g. black), otherwise the maximum value is assigned to the pixel (e.g. white).

The most trivial way of thresholding is global thresholding. It is done by specifying a global threshold value and comparing each pixel of the image with this value. However, this approach proves not to be consistent in most cases, as light distribution on the image is usually uneven – one part of the image is lighted, while another part remains dark. Therefore the best approach would be so called adaptive thresholding, which would use separately calculated threshold values by statistically checking the intensity values of the of the surrounding pixels of each pixel in specified area of the image.

As a result, different threshold values are calculated for different regions of the image and this gives better results for images with varying illumination. The stage of thresholding is an important stage of preprocessing before OCR, as it creates black-and-white image ready to serve as input for recognition by the OCR engine.

## Optical character recognition

Optical character recognition, or simply OCR, is the machine recognition of printed or handwritten characters from an image. As a rule, OCR systems can recognize text with different fonts, both typewriter and computer printed characters, but some complex OCR systems can even recognize handwritten text. (http://www.pcmag.com/encyclopedia/term/48267/ocr)

There are various approaches used for OCR engines design:

* Matrix matching: Each charecter of the image is represented as a pattern and is compared to stored glyphs. Such kind of recognition is not suitable in case text with different fonts can be found in the same image.
* Feature extraction: Each character is represented by a set of features (height, width, density, lines, loops etc.). The absence or presence of some exact features is used to determine the character.
* Neural Networks: The main idea of this approach is to train an artificail neural network with a set of training data input, which would be used for the further recognition.

(<http://www.cisjournal.org/journalofcomputing/archive/vol4no6/vol4no6_2.pdf> )

# System design consideration

## Choice of the architecture

Before the start of the implementation the author pointed out 3 possible ways of implementing the mobile application, all having their pros and cons:

1. Develop a mobile application with all the functionality and business logic in it. In such case there is no server side at all. All the stages of images preprocessing before OCR, OCR itself, expense information extraction from the OCR result as well as storage of the user expenses is done on the device where the application is installed.

Pros of this solution:

1. No need to implement the server side and thus it is to possible to avoid additional complexity in the development process. There will be no need to design server-based authentication mechanism, session handling and data retrieval logic because all the functionality would be in the mobile application itself and all the data would be stored locally on the device.
2. The mobile application does not need internet connection, as all the functionality of it would be available offline.

Cons of the solution:

1. As all the user’s expense entries are stored on the device only, there is no way for the user using multiple devices to have a single account with all of his expenses present on all the devices automatically. In order to equally keep track on all of his expenses on multiple devices, user would have to manually duplicate the expense entries, which may be very inconvenient.
2. For example, if the device storing all user’s expenses data becomes unusable or if a restore of device’s factory settings is done, then all the user’s expenses data would get lost, which is definitely an example of bad user experience.
3. As all the receipt image preprocessing stages and OCR are performed on the device, it implies that the application will have to include dependencies for both image preprocessing and OCR libraries, which will make application size much bigger (according to the author’s estimations, more than 60 MB). Large application size may prevent users from downloading and installing it.
4. If in the future there is a plan to introduce support for the other mobile platforms besides Android (e.g. iOS, Windows Phone), there might occur problems, as there is no guarantee that the libraries for image preprocessing and OCR used in Android can be easily integrated into the application for other platforms in the same way.
5. Develop a mobile application (client) and a server the mobile client would communicate with, however in this case the client is still “fat”, i.e. it still contains a large part of the business logic. The process of image preprocessing before OCR, OCR and expense information extraction from OCR result would be still performed on the client side, while the server would be used for storing and retrieving expense information.

Pros of the solution:

1. User may have a single account and use it with numerous devices and all the expenses data is synchronized with the server. There is no threat of losing the expense data as in case of the previous solution, as all the data will be stored in the database on the server side.
2. As the receipt image preprocessing and OCR with the following expense information extraction takes places on the client side, there is no need to push the whole image to the server in order to make expense extraction from that. Instead of that, only the extracted expense information itself is posted to the server.

Such kind of approach acts in favor of decreasing the internet traffic intensity,

as the payload size for the post request with the expense data is around several hundred bytes instead of several megabytes in case of the whole image.

Cons of the solution: same as the points c) and d) of the 1st solution.

1. Develop a mobile application (client) and server. The mobile client should be thin and all the business logic should be located on the server side. The expense displayed to the user are retrieved with a request to the server, the same goes for submissions of receipt images for OCR and further expense information extraction.

Pros of this solution:

1. Same as the point a) of the 2nd solution
2. The client side is thinner, as it does not include libraries for the image preprocessing and OCR. As a result, the application size is smaller (< 20 MB), making it more attractive for the user.
3. The whole application is more consistent - if there are client applications for multiple platforms, then expense information extraction from receipt works in the same way for all the devices.
4. Expense information extraction from receipt image techniques may need some improvements in the future. In order make some improvement in receipt image preprocessing stage, or to introduce an upgrade with an aim to increase OCR accuracy, there will be a need to make considerable redesign of the application. However, the server side redesign turns out to be less complex, as there will be no need to make changes for all the client applications for different platforms. In fact, the client side will continue communicating with the server using the same HTTP resource API, and the end-user will not have to take care about installing the latest client application updates in order to experience the better quality of the software.

Cons of the solution:

1. The need to send the whole receipt images to the server side for processing increases the load on the internet traffic. Furthermore, it is also more time consuming in comparison with on-device processing.

Taking into consideration the pros and cons of the above mentioned possibilities of the architectural choices, the author had generally to decide whether to go on with either the 2nd or 3rd approach. The reason why the author did not consider the 1st solution at all is mainly because it had more cons than pros and particularly due to the cons point a) and b), as it contradicts such aspects of the application as reliability and usability from the user’s perspective.

Both 2nd and 3rd architectural solution had their considerable pros and cons, and it was hard to definitely determine, which of them would help to meet the author’s need in the best way. In the end, the author decided to use the 3rd solution in his application, particularly because of it’s pros points b) and d).

## System key components design overview

The application was decided to be implemented in the following way: a mobile application represents the client side and communicates with the server by making requests and receiving corresponding responses: The server should contain all the business logic regarding the the expense recognition from the receipt image as well as have functionality for reading and representing user’s expenses in a form, which is understandable for the client side.

In order to proceed with the implementation of the application, decision regarding the system structure, behavior, data exchange protocols, data exchange formats as well as data storage options should be made.

### Mobile application client: native, web or hybrid?

Before starting implementation of the mobile application it is necessary to decide whether to build it as a native, web or a hybrid application.

#### Native application

A native mobile application is written using the language, which is specific for the device’s platform (for instance Swift for iOS, Java for Android), targeting the API and features of the selected platform by using a platform-specific SDK. As a rule, a native application in downloaded from a platform-specific app store and then installed directly on the device.

The benefits of using the native application development approach are possibility to use the variety of technical features provided by the platform API, accessing device’s hardware features as well as using platform-specific visual design elements for the purpose of creating an application with native look. Furthermore, native applications tend to have better performance in comparison with web-based solutions.

From the perspective of a software developer, making a native application requires knowledge of platform-specific language and API. In order to create and maintain a native application, which can function on a large variety of devices, one must know and consider all the possibilities and constraints of the API-s as well as the possible devices’ hardware limitations.

#### Web mobile application

A web mobile application is another widespread mobile application type. Such kind of application is generally a web-page, which is accessible through the web browser of the device. Such kind of application is created using HTML, CSS and JavaScript and sent to the user from the server using the Internet. At the same time, mobile web application’s design must be adapted for the size of the target mobile devices.

The are numerous benefits of developing a mobile application as web application. Firstly, familiar web technologies are used for that, so it does not require knowledge of any platform-specific programming language or API, and as a result, it becomes possible to release a cross-platform application with less effort and investments in comparison with the native approach. Secondly, in order to fix a bug or introduce a new feature in the application, there will be no need to make users download the application update – a change made on the server side hosting the mobile application will be seen by users automatically with a request to the server running the updated application.

At the same time, there are still some disadvantages of choosing the web mobile application approach. First of all, there is lack of support of distinct features by some web browsers. Being not aware of this fact may result in designing an application, that will not function or look as expected in some web browsers. Furthermore, web applications have limited access to some device’s platform-specific or hardware features. For example, it is impossible to make a web application on Android process incoming push notifications or access device’s notification manager. Though, for example HTML5 allows invoking of the device’s camera. Mobile web applications also have poor support for working offline.

#### Hybrid mobile application

Hybrid mobile application may be regarded as a compromise solution between the native and the web approach. It can be described as native wrapper, which is used for showing application web views and the same time has possibility to use all native app features. The describe effect is achieved with the use of special frameworks, such as Apache Cordova. Consequently, it combines such pros as shared HTML, CSS and JavaScript from the web approach and ability to use platform-specific features and access device’s hardware from the native approach. It still has some drawbacks, such as no platform native look and feel for the shared web view as well as lack of responsiveness in comparison with the native applications.

#### Choice of the mobile client designing approach

The author had to make a choice mainly between implementing a native or a hybrid application. The pure web approach did not meet the needs of the author, as the application being designed may require usage of device’s notification manager, which is not available in the context of a mobile web application. Moreover, implementation of offline working possibility may also be a problem in case of the web approach.

Releasing the application for multiple platforms with least effort was not among the goals of the the thesis, and in case of a hybrid application there will still be a need to write some part of native code, so the author decided to choose the native approach, to some extent due to better performance of the native application.

### Images processing flow

#### Initial synchronous solution

In the beginning, when the author designed the initial solution for receipt image processing (images preprocessing, OCR, expense information extraction), it was fully synchronous. Synchronous means that a receipt image was posted to the server with a HTTP request from the client side, which remained waiting for the response with the extracted receipt data until it arrived. On the client side the response handling was designed in an asynchronous way, i.e. waiting for the HTTP response did not block any user’s actions within the application, and the reponse was handled as soon as it arrived.

Though from the client side perspective everything seemed to work correctly, such kind of approach was not suitable, mainly because the total time between the sending of request and receiving the response was ~15 seconds, and it was too long from the point of the server throughput. As a server has a finite number of worker threads (for example, for Tomcat 7.0 server the default number is 200) used for handling users’ requests, in case request handling takes as long as 15 or more seconds, there exists a threat that in case of many simultaneous users’ requests the server may run out of idle working threads ready to process incoming requests. In such case the requests will have to wait for some worker thread to become free for processing, and too long waiting may result in request timeout. From the point of the user experience, receiving timeouts is not a good practice, so the author considered to find out better solution instead of the described synchronous approach.

#### Asynchronous solution

To find out the best approach for the designing the asynchronous processing solution, it was first necessary to determine, which stage of the process was the so called “bottleneck”. After investigation it turned out, that the request processing timeline propagated in the following way: the image uploading stage took 3-5 seconds on average with the 3G Internet, while the stages of image preprocessing before OCR and OCR itself took around 8-10 seconds. This means that after a user uploaded a receipt image, the response considering successful image upload (or an error response in case of erroneous scenario) could be sent to the him, while the image preprocessing as well as OCR and further expense information extraction could be taken out from the request-response cycle and done asynchronously, i.e. when the server has corresponding resources to perform these stages. Such approach can improve server throughput and increase number of idle worker threads ready to process arriving requests and, consequently¸ help to get rid of request timeouts.

After a receipt image is uploaded to the server by the user for further processing, it must be put into a queue, which could be of FIFO type, and a dispatcher mechanism could take awaiting images for processing from that queue. The database was decided to be used as a queue, as it preservers all the images even in case of a server fallback as well as makes it easy to retrieve the pending images in the right order.

Processing itself should be better performed for multiple images concurrently to increase the overall processing speed. As soon as the expense information is extracted, it should be stored in the database.

#### Notifying the user about image processing result

As the receipt image processing is now organized in an asynchronous way, the client side cannot exactly determine whether processing of the image has ended or not. Thus, in order to find out whether the result is available, it is necessary to poll the server for the result by sending request to get the result. If the result is not yet available, the request must be repeated after a distinct timeout. Such kind of approach though seems not to be good from the perspective of the server – numerous users will be spamming the server with requests, and it will create additional load on that.

That is why it should be better to apply the Hollywood principle, stating “don't call us, we'll call you.” and favor push notifications over polling in order to decrease the load. As soon as the image is processed, a message with a processing result will be pushed to the user’s device. For the purpose of sending push notifications cloud services like Google Cloud Messaging for Android should be used. A good feature of a push notification is that it does not require the application to be running in order to reach the target device, so the user, who has uploaded the receipt image will receive the result notification even if he has closed the application after uploading the image. The device will receive a wake-up event as soon as the notification arrives, and notification event will be passed to a corresponding service, and user will be able to see the extracted expense, or error in case recognition was not successful.

### Data exchange format between client and server

JSON was chosen to be the format for interchanging the data between the client and the server side, as it is lightweight and less verbose in comparison with XML. What is more, in recent years JSON has become a standard for the newer generation of web applications.

### Authentication and authorization

To use the functionality of the application, the user must be authenticated and granted corresponding rights. When a user passes the authentication stage, a session for him is created. Logged in user data (Display name, e-mail) is then stored locally on the user’s device. For the purpose of authentication the author decided to use Google APIs, as it should be comfortable from the user’s perspective and there will be no need to create extra an account and memorize the password for that.

### Offline data storage

If user has once logged in, later he has opportunity to capture images of the receipts while being offline. When the user gets the internet connection back, he will be able to send the images to the server. The captured images must be stored on the device, and a suitable place for them is a on-device database, as it will make easy to retrieve the images in case the device is used by multiple users, and every user wants to see in the application the images captured by him.

# Technologies

This paragraph contains description of technologies, which were used for the development of both server and client part of the application. In each subparagraph there is description of corresponding technology and a short remark saying where this technology was applied in the application. More precise description of how the technologies were applied can be found under the “Implementation” point.

## Choice of platform and programming language

Before implementation the author had to choose the platform and language of implementation for server and client side.

The best candidates for implementing the server were Java, Ruby and Python, as these are considered to be the most mature platforms for designing a REST service. Due to the fact that the server side will have to deal with compute-intensive tasks and concurrency, the author decided to go for Java, as it provides better performance in such cases.

As it was decided to design a native client application, the platforms the author had to choose from were Android, iOS and Windows Phone. As the author had neither any experience of programming in Swift or C#, nor device for testing an iOS or Windows Phone application, the iOS and Windows Phone platforms were not suitable. As for Android, the author had also almost no experience of developing for this platform, though he had knowledge of Java programming language and a device which could be used for testing purposes. As a result, Android was chosen as a platform for client application.

## Spring framework

Spring is a Java-based development framework created with the purpose of enhancing and development of applications for the Java platform. It provides comprehensive programming and configuration model for Java-based enterprise applications. The main emphasis of the Spring framework is still put on the web applications.

Spring framework is used on the server side of the application with the following purposes:

* Inversion of control, or dependency injection approach, allowing to create more loosely coupled and reusable component, which are the building blocks for the application.
* Design of the REST service controller layer, which maps URI resources to corresponding actions on the server side.

In the described project the author used Spring Boot, which is a sub-project of the Spring framework. The main advantages of Spring Boot is that it favors convention over configuration, i.e. there is no need to explicitly define configuration of the application, unless configuration different from the default one is required. Moreover, there is an opportunity to run an application as a .jar on an embedded server (Tomcat 8.0 in this project) instead of deploying a .war file in the container.

## Spring Security

Spring security is a framework for Java-based applications which provides mechanisms for user authentication and access control. It makes easy to create custom means of authentication and integrate them into application’s authentication flow. Configurations regarding the rights granted to the authenticated user as well as handling of exceptional cases can be done directly in the Java code in a declarative way.

In the application the author develops Spring Security is used on the server side for user authentication and authorization.

## Hibernate ORM

Hibernate ORM is an object-relational mapping solution, created for Java programming languages. It is used for the purpose of mapping data from an object model representation to a relational data model representation, i.e. mapping Java classes to corresponding database tables. Based on this mapping, Hibernate framework generates SQL queries for data retrieval and takes care of transforming the result set returned by the database query into Java objects.

It also provides similar techniques for insertion, update or deletion the database entries corresponding to a Java object.

The mapping of a Java class to database table, ot a set of joined tables, is done either in XML configuration files or with the use of Java annotations in the class being mapped. Such approach prevents a developer from writing extra SQL queries as well as boilerplate code for mapping the queries’ result sets to entities.

In this project Hibernate ORM is used on the server side for performing CRUD operations. It is also worth pointing out that the author did not use Hibernate ORM directly, but rather used Java Persistence API (JPA) and Hibernate ORM served as an implementation of the JPA specification in this case. Such approach favors programming using interface rather than implementation and therefore is making components more loosely coupled, i.e. Hibernate ORM can be replaced with any other Java object-relational mapping framework corresponding to the JPA specification without any significant changes in the Java code.

## PostgreSQL

PostgreSQL is one of the most powerful object-relational database systems, which is fully open source. In the current project, PostgreSQL is used for storing users’ expenses and receipt images related data on the server side. As it was mentioned above, to perform CRUD operations JPA with Hibernate ORM implementation were used. Hibernate ORM uses Java Database Connectivity (JDBC) to perform the database reading and writing operations.

## Jackson

Jackson is a Java library providing a set of data-processing tools with the main focus on JSON parsing an generation, as well as binding JSON with Java POJOs.

In the current project Jackson library found its application in the controller layer of the server. There it is used for transforming incoming POST HTTP requests body to Java objects, as well as for creating the HTTP JSON response body from Java objects.

## Dagger

Dagger is a library used with the purpose of dependency injection. It was created with the purpose of reducing the amount of boilerplate code (Factory pattern etc.). It provides simple mechanism for declaring dependencies providing logic as well as satisfaction of dependencies using Java annotations. As a result, an application is built of interchangeable, reusable and loosely coupled components.

In the current project Dagger dependency injection tool was used for development of the Android client-side of the application.

## Retrofit

Retrofit is a Java library (mainly targeting Android) providing tools for implementing a REST client for sending HTTP requests to the REST API of the server. By using Retrofit it is possible to design service objects, the methods of which are mapped to the corresponding server URI resources for distinct HTTP requests. The library has support for both synchronous as well as asynchronous tasks and makes it easy to transform HTTP JSON responses to Java objects.

In the current project, Retrofit was used in the Android client-side of the application for communicating with the server via HTTP.

## OpenCV

OpenCV (Open Source Computer Vision Library) is a computer vision and machine learning software library. The library has more than 2500 optimized algorithms, providing, for example, functionality of detection and recognition of faces, motion recognition, extraction of 3D models of objects, finding similar images etc.

The library itself is written in C++ programming language, but it also has a Java which uses Java Native Access (JNA ) for purpose of invoking the native code.

The library has a set of comprehensive tools for image processing, allowing to perform the operation with changing color spaces, geometric image transformations, thresholding operations as well as morphological transformations. Mainly because of these functions this library became a suitable candidate for performing image preprocessing on the server side of the current application.

## Tesseract OCR

Tesseract OCR is considered to be the most accurate open-source engine for optical character recognition. It proved to be working on Windows, Linux and Mac OS X. Though possibility of compiling Tesseract OCR for Android and iOS exists, these platforms are not considered to be well-tested platforms.

Tesseract OCR uses a two-pass recognition process, which involves machine learning. During the first pass it performs an attempt to recognize each word from the image. Each word which satisfies a set of distinct conditions is passed to the adaptive character classifier in order to use it for further recognition. Considering the fact that the adaptive classifier may have learnt useful features only when it reached the bottom of the page, a second run over the page is executed. During the second run there is high probability that the trained mechanism will recognized the words that were not recognized during the first run.

It is also worth mentioning that Tesseract OCR can perform recognition from an image with a minor skew of the text. This means that no explicit deskewing operation must be performed on the image, therefore the quality of the image would not suffer.

Since version 2.00 Tesseract OCR is fully UTF-8 encoding capable. Tesseract OCR is written in C++, but a Java wrapper called Tess4j can be used for Java projects.

In the current project, Tesseract OCR was used by the server side of application for converting the receipt image to text.

# Implementation

This paragraph covers the implementation process of the application, explaining in details the most important stages of process and explaining usage of the above mentioned technologies and design solutions. The client and the server side development process is covered together, because these component were implemented in parallel and are closely interconnected with each other. The source code of the application can be found here:

<https://github.com/stolzzz/tess-server> - server

<https://github.com/stolzzz/tess-client/> - client

## Authentication

For the purpose of authentication Google Play Services were used. It is a good choice from user’s point as it does not require any additional account creation, and mainly all the Android users have a Google account. Furthermore, this solution may be considered quite mature, as it is widely used in modern applications.

Google authentication is based on OAuth 2.0 protocol, allowing granting a third-party limited access to the protected resources of the user without a need to pass user credentials to this third-party.

### Prerequisites for using authentication with Google

In order to start using authentication with Google Play Services, it was necessary to register the Android application in the Google Developers Console by providing the package name of the application (com.roman.ttu.client) as well as SHA1 fingerprint of the key store to be used for signing the release APK of the application. A unique Client ID for the application was also generated then. After registration the application can make call to the Google Play Services API and in order to do that Google Play Services library must be included to the Android application.

### Client side token retrieval flow

The entry point for the Android application is StartActivity class. In onCreate() method of this activity an account picker is invoked in case no user was logged in by that moment. The picker allows to choose a Google account the user wants to proceed using application with. After account is selected, a call to Google Play Service API in order to receive the access token is made. The arguments provided to the call are the e-mail corresponding to selected account and the OAuth scope, which determines, which kind of the user’s data may be accessed by the party using the access token. In this application the scope is oauth2:https://www.googleapis.com/auth/userinfo.profile, because the only information the application currently needs to know about the client is his Google user id and his display name.

The access token is retrieved with a synchronous transaction over the Internet, that is why it needs to be executed by a separated worker thread to prevent blocking of application’s UI. For this purpose a subclass of android.os.AsyncTask was introduced.

### Token transmission security prerequsites

After the access token arrives to the client side, it must be sent to the server for further validation and if it is successful, then a session for the user is initiated. The access token is transmitted from the client side to the server over HTTPS in order to provide secure communication. In order to prepare the server to accept HTTPS connections, an SSL certificate is required.

As currently the goal of the project is not to release an application directly to run in production environment, the author did not go for obtaining a certificate issued by a known certificate authority (CA) as it is not free of charge, and created a self-signed certificate for testing purposes. In order to generate a certificate to be used by the server a command-line program named *keytool* was used. The certificate was generated with the use of RSA algorithm for creating the key pair, each key with the size of 2048 bits and PKCS12 was used as the key store type.

In order the Android application could check that the self-signed certificate of the server is a trusted one, a custom javax.net.ssl.TrustManager implementation had to be introduced. This implementation explicitly performs a check whether the certificate of the server the client side is performing a handshake with is exactly the certificate that the client side trusts. If a certificate used by the server has the same Common Name (CN) and it’s public key is equal to the trusted one’s, then this certificate proves to be trusted.

### Server side authentication

The author has chosen Spring framework for implementing the web application. Spring framework allows to implement custom javax.servlet.Filter objects, which can intercept and filter requests before they reach the target resource of the server.

A custom implementation of an authentication filter was introduced by the author. It performs the following check – if a user is not authenticated, then a try to retrieve the access token from request is made. If the token is not found from the request, then an exception is thrown. If the access token is found from the requests, then authentication with the provided token is initiated.

Authentication is executed by a class from Spring Security module name org.springframework.security.authentication.AuthenticationManager. This class uses implementation of org.springframework.security.authentication.AuthenticationProvider to perform the authentication process. The author designed a custom TokenAuthenticationProvider for this purpose. Internally TokenAuthenticationProvider first of all makes a call to the Google API in order to retrieve the acces token information in JSON format. If such token was not found by Google, then authentication fails and a corresponding exception is thrown. In case token information arrives, the server makes a check whether the token was issued for application with the required Client ID (it was generated when the application was registered in Google Developers Console). If the Client ID, which arrived with the response from Google matches with the expected one, then Google user id and display name of the user is obtained from Google API by the server, otherwise the authentication process is interrupted. In the end, the session for the user is initiated. User id and name become accessible on the server side using SecurityContextHolder.getContext().getAuthentication().getPrincipal(). Session expiry time is transferred to the client side in HTTP response headers.

### Client side session handling

Android client side keeps track on the state of the session. It persists locally the time when then session is to expire, and it is updated with each response received from the server. Activities, which perform calls to the server, or authentication-aware activities, check whether the session has expired, and if it has, perform the authentication process. In case of successful authentication, the application usage flow which might have been interrupted by authentication, continues.

## Receipt image processing and OCR

A users captures the image of the receipt with the use of device’s camera. After an image is captured and there is internet connection available, the image is sent to the server for further processing. An image is encoded with *Base64* encoding in order to be sent as a JSON string. When an image reaches the server, it is placed into a processing queue, represented by a database table named *receipt\_picture\_data*, mapped to a Java class ReceiptImageWrapper. A HTTP response is sent to the user as soon as the images are saved to the database.

As it was previously mentioned in the paragraph 3.2.2, image processing (preprocessing and OCR) is designed to be executed asynchronously, and this occurs as soon as there is free resource to do processing, i.e. there are free worker threads to do that. A dispatcher thread retrieves a batch of images for processing from the database, retrieving the oldest images first. These images are then passed to a number of threads, which process them in parallel. A number of threads in the thread pool to perform processing is N + 1, where N is the number of CPU-s on the machine. Such number of threads is chosen, because image processing is a compute-intensive task, and with such number of threads the system achieves optimum utilization. When there are no images to process in the database, the dispatcher thread is put to sleep for 5 seconds and after that polls the database for the images again.

As Hibernate ORM is used for CRUD operations in the databases, images are retrieved from the database using HQL (Hibernate Query Language), which is similar in appearance to SQL, but the syntax of its queries is based on the level of Java objects corresponding to the database tables.

During processing the image is first decoded from Base64 string and is written to file. Then preprocessing of the image takes place. During preprocessing operations of conversion to grayscale, denoising, morphological closing and adaptive thresholding are executed on the image using corresponding methods provided by OpenCV library. The point of all these operations is to prepare a better image for OCR and the meaning of all of them was described in the paragraph 2.1.

After preprocessing the image is passed to Tesseract OCR engine to extract the text from the image. It is done simply by creating a new instance of Tesseract1 object and calling its method doOcr(imageFile), where imageFile is a file with a preprocessed image.

## Receipt information extraction from OCR result

As a result of OCR a string representing the recognized text is returned. From this result it necessary to extract the name of the merchant the purchase was made from and the total sum of the purchase.

If one takes a look on the variety of the receipts issued by different Estonian enterprises, then it is clear that there is no any common standard or pattern of how information on the receipt is organized and represented – each enterprise does it the own way. This makes very hard the extraction of the required information, as the solution responsible for information extraction must consider different formats of information representation.

### Enterprise name extraction

When it comes to the location of the enterprise name of the receipt, then usually it can be found in the “header” part of the receipt. However, there no any exact rule that name of the enterprise is, for example, is always located on the first line on the receipt or somewhere else, so it is impossible to locate it without knowing the format of the receipt of an exact company. Furthermore, there is a chance that during OCR some letter in the enterprise name may be recognized in a wrong way, and such case would need special handling in order to restore the original name. An alternative and more reliable solution for retrieving the enterprise name through enterprise registration number must be used.

Enterprise registration number is usually provided in the “header” part of a receipt, preceded by “reg.nr.” or “reg.kood” string, which makes it simple to locate and parse that. An enterprise registration number itself is a unique number consisting of 8 digits, which is issued by the Estonian business register to an enterprise. If parsing the registration number succeeds, it is possible to retrieve enterprise name (as well as other basic enterprise information) using the Estonian e-business register web service, which is using SOAP standard. Unfortunately, the service is not that simply accessible – it requires signing an agreement for using the service as well as paying for its usage. That is why the author implemented a simple web scraper targeting e-business register web page for the testing purposes. It sends a request for getting the enterprise basic information for the supplied registration number, receives HTML in response and parses the enterprise name from that. In order to minimize the number of requests to the e-business register, an application-level cache, caching key – value pairs of registration number and enterprise name, was introduced.

### Total sum extraction

The total sum entry of a receipt is usually located in its bottom part. As the result of OCR persist the original line endings, it is possible to determine where a line starts or ends. For total sum extraction the following logic was used: the lowermost line, starting with a keyword meaning the total sum (“kokku”, “summa”, “maksta”, “tasuda”, “vahesumma”) must be found, and the rightmost decimal number with two fraction digits in this line is the total sum. There might be several lines starting with the mentioned keywords, but the lowermost is chosen, as unlike the others, it represents the final sum the customer has to pay, with all the taxes and discounts applied.

### Technique of extraction

In the beginning, the author implemented extraction with simple string parsing with the use of regular expressions. However, in some cases this technique proved to be not be working. The OCR engine may have recognized some of the characters in the wrong way, and as a result, some of the recognized words may be misspelled. That is why trying to find an exact match for the strings is not suitable in this case. In order to solve this problems approximate string matching must be considered. For this purpose FREJ (Fuzzy Regular Expressions for Java) library was used. FREJ allows to create regular expressions (the syntax is though slightly different in comparison with usual regular expressions) used for approximate string matching (i.e. there may exists a defined number of errors for a word to still be considered matching to the specified pattern), and use it to find the required keywords. It is possible to define how strict the matching should be specifying the threshold value. The threshold value is a double starting from 0.0, meaning that an exact match is required and ending with 1.0, meaning that it would match any string. In this project the default value of 0.34 was used. For example in such case, if the word “summa” was recognized as “sunma” by the OCR engine, it is still considered to match FREJ regular expression.

## Notifying the user about a processed expense

In case both enterprise name and total sum were successfully extracted, the extraction is considered to be successful, otherwise it is erroneous. If an extraction is successful, the extracted expense is persisted in the database in an initial state, i.e. it is still need to be explicitly confirmed (or declined, in case, for example, wrong total sum was recognized) by the user. The user is notified about the result in case of both successful and erroneous recognition.

User is notified about the result of receipt image processing with a push notification. For this purpose Google Cloud Messaging is used.

### Client-side notifications handling

In order to integrate Google Cloud Messaging into the client part of the application, Google Play Sevices library must be present among the dependencies of the Android application. An application must be also registered in the Google Developers Console. The Android application manifest (AndroidManifest.xml) must be modified in order to allow receiving Google Cloud messages, accessing network in order to send Google Cloud registration ID of the device to the application server and keeping processor from sleeping in case a message arrives.

Before the Android application can receive cloud notifications, it is required to obtain a registration ID from Google Cloud messaging service. This is a unique ID, which specifies a distinct application on a distinct device. Registration ID is obtained after the first authentication and is then persisted locally on the device. This registration ID is then always sent along with the receipt image to the server and serves as an address where a notification with the expense extraction result should be sent.

The Android application must be explicitly subscribed to receive the notifications. For this purpose an implementation of a WakefulBroadCastReceiver was introduced. In case a cloud message arrives, it receives the message and passes it to the corresponding service for further handling. The service decides whether a message contains successful expense extraction result or an error and shoots a corresponding notification using Android NotificationManager. The user can see the arrived notifications in the notification area and press them in order to get information concerning the result. In case of a successful result, a confirmation activity for an expense, allowing to either to confirm or decline an expense, appears. In case of an error, an activity with an error message and faulty extracted expense information is shown.

### Server side notifications handling

The server sends an expense information or an error to the Google Cloud over HTTP protocol in JSON format. It adds the registration ID which arrived with the receipt image in order to specify the device which will receive the notification. For the purpose of authorization, the server provides the Google API Key in the header of the request, which was specially generated for the server part of the application in Google Developers Console. The expense information is then pushed through the Google Cloud to the target device of the user.

## Displaying user expenses

The application user can see a list of his expenses for the specified period and a total sum for all his expenses for this period. In case of many expenses (more than 15), only the first 15 latest expenses are displayed at first in the expense list, and as users scrolls the list down, the expenses are retrieved in batches containing 15 expenses. The expenses shown in the list are both confirmed and unconfirmed. The unconfirmed expenses are not considered when calculating the total sum of the expenses for the period. The unconfirmed expenses can be confirmed or declined directly in the expenses list.

## Working offline functionality

In case there is no Internet connection, the user can take receipt pictures offline and send them to the server as soon as the device is online. The captured receipt image is stored as *Base64*-encoded strings in an on-device SQLite database, which is provided by Android SDK. This makes it easy to retrieve the images waiting to be sent for a distinct user, who is using the application at the moment. The preview list of pending images is displayed to the user using Android ViewPager. If a pending image is successfully sent to the server for expense extraction, it is then removed from the on-device database.

## Expense manual submission

The application has an opportunity to submit an expense manually in case of extraction of the information from the receipt is failing due to poor quality of a receipt or some other reason. It requires submitting the registration number of the enterprise and the total sum of the expense to the server. The enterprise name in such is retrieved using the enterprise registration number in the same way as in case of recognition of expense from a receipt.

# Implemented application

## Testing expense extraction from receipt functionality

### Testing process

The application’s functionality of expense extraction was tested by the author throughout the development process of the application. Receipts of 20 Estonian enterprises in different condition – new, crumpled and worn-out, were used for the testing purposes. LG Nexus 4 smartphone (released in 2012) with and 8-megapixel camera running Android 5.1 operating system was used as a testing device.

Before the image preprocessing and approximate matching of the expense from OCR result stages was implemented, the recognition rate was relatively low – expense extraction was successful on average 2.4 times out of 10. This result was received after a series of 10 attempts to extract expense from each of the 20 receipts.

After the stages of image preprocessing and approximate matching were implemented, the same series of attempts was repeated, and now the result was on average 6.2 out of 10 successful recognition. It proved that the preprocessing stage as well as fuzzy expense extraction were a useful enhancement for imptoving the recognition.

### Analysis of the testing result

The recognition rate of 6.2 out of 10 may be considered quite good in terms of the fact that printed text on the receipt is not of ideal quality, and the image might be blurry and not lighted enough. Despite this, the investigation was made to make clear the main reasons, why recognition was failing. In around half of the failure cases either enterprise registration number or total sum was still recognized correctly. Firstly, lower recognition rate was detected for receipts, which have the total sum label printed with a font (usually large and bold), which significantly differs from the font of all the other text of the receipt. This can be explained in a way that the OCR engine’s algorithm cannot adapt to the different font, which appears only one or two times on a receipt. Secondly, recogniton was invalid in some cases beacuse digits in either total sum or enterprise registration number, which may be visually similar (e.g. “5” and “6”) got confused, which points to the fact that the receipt image resolution was not high enough for OCR.

## Estimation of the selected technology stack

The Spring Framework, Spring Security and Hibernate ORM proved to be a justified choice for implementing the server side, as they helped to design a loosely coupled as well as easily configurable piece of software.

PostgreSQL database proved to be working as a queue for asynchronous processing of images, though from the perspective of future development an alternative of using a message queue (such as ActiveMQ) may be considered, as it is designed specifically for such purposes. The message queue uses push strategy instead of pull, so there will be no need to poll the database. Moreover, a message queue is considered to be better from the point of scaling.

Retrofit library used for the Android part made easy to design reusable components for communicating with the server without any need to write boilerplate code, especially for handling the erroneous cases.

Dagger library, which was also used for the Android part helped to design logic for dependency injection of required components and focus on writing the code which contatins business logic.

OpenCV proved to be a suitable tool for image preprocessing before the OCR. Its usage helped to increase the expense recognition rate significantly. The library has comprehensive documentation, which was very helpful from the point of choosing the suitable methods and parameters for image preprocessing.

Tesseract OCR engine can also be considered as a suitable tool for performing OCR. Though, in order to achieve the best results with it, images of high quality must be provided as an input, which is not completely possible in a context of the current application.

## Possible improvements

The improvement may be made in a sense of adding new functionality to the application, as well as improving the existing part. For example, the function of expense recognition from a receipt may be improved in a way of introducing machine learning for the purposes of correcting recognition errors relying on the history of successfully recognized expenses from receipts of distinct enterprises.

# Summary

The aim of the thesis is to create a mobile application for the purpose of expense tracking. The application should be able to capture images of receipts issued by Estonian enterprises, automatically extract expense information from them and store it in the users’ expense lists. The application should also provide secure means of authentication as well as possibility to see user’s own expenses for a specified period.

As a result of the work, a mobile application for Android corresponding to all the above mentioned requirements, was implemented. All the functionality of the application, including receipt image preprocessing, OCR from the preprocessed receipt image and expense information extraction from OCR result was designed with a set of open-source technologies, meaning that such kind of application does not definitely require using proprietary software solutions. The design of the system allows futher simple integration for other mobile platforms besides Android, as they will be using the same REST API provided by ther server side.

The author analyzed the main possible ways of organizing the architecture of the application as well as described the design of the separate components of the system and their interaction. The main arguments in favor of choosing the client-server architeture of the software, native mobile application implementation approach as well as asynchronous flow of receipt images processing were pointed out and compared with the possible alternatives.

It is possible to say, that the goal of the thesis was achieved, as the designed software works as it was expected. Despite this fact, there is still possibility for further improvements. Though from the perspective of expense extraction from a receipt image the application showed relatively good results, it is possible to conclude that it is hard to design a totally reliable solution, which would work in 100% of cases. Such factors as the image quality and state of the receipt influence the final result of expense extraction signigicantly, and it is impossible to completely eliminate the influence of these factors solely by using even advanced image processing mechanisms. That is why in oder to improve the recognition rate introducing a machine learning mechanism, which woulde be trained with successfully recognized receipts, could be the next goal for the future.

# Kokkuvõte

Töö eesmärgiks oli luua mobiilirakendus, mis on mõeldud kasutaja kulude pidamiseks ja dokumenteerimiseks. Mobiilirakendus peaks võimaldama teha pilte Eesti ettevõtete poolt väljastatud ostutšekkidest, automaatselt võtta välja informatiooni kulude kohta nende tšekkipiltide pealt ja pärast lisada saadud informatisooni kasutaja kulude listi. Peale seda, rakendus peaks võimaladama kasutaja turvalist autentimist ning pakkuma võimalust näha kasutaja kulusid ette antud ajaperioodil.

Töö tulemusena sai implementeeridut mobiilirakendus Android platvormi jaoks, mis vastab kõigile üleval mainitud nõuetele. Kogu rakenduse funktionaalsus, kaasa arvatud osttutšekkide piltide eeltöötlemine, optiline tähtede tuvastus eeltöödeldud ostutšekki pildi pealt ning tuvastatud tulemusest vajalikku kulu informatsiooni leidmine, oli realiseeritud kasutades avatud lähtekoodiga tehnoloogiaid, mis omakorda tähendab, et taolise rakenduse loomiseks ei pea ilmtingimata kasutama proprietaarseid tarkvarakomponente. Süsteemi disain teeb võimalikuks luua kliendirakendusi peale Androidi ka teiste mobiiliplatvormide jaoks, kuna need hakkavad kasutama sama serveri REST API-d.

Töö autor autor analüüsis võimalikke viise kuidas organiseerida rakenduse arhitektuuri ning kirjeldas eraldi ka erinevate tarvarakomponentide disaini ning nende komponentide koostoimet. Toodi välja põhilisi poolt argumente klient-server arhitektuuri valiku, põlisrakenuse arenduse ning asünkroonse ostutšekkipiltide töötlemise kohta ning samas võrreldi neid lähenemisi võimalikke alternatiividega.

Saab väita, et töö eesmärk on saavutatud, kuna valmis saadud rakendus töötab nii, nagu oli eeldatud. Vaatamata sellele, saab välja tuua parandusi, mida võiks tulevikus realiseerida. Kuigi kulu informatiooni ostutšekki pealt väljalugemise funkntsionaalsus toimib suhteliselt hästi, saab järeldada, et on väga keeruline implementeedida seda funktsionaalsust nii, et see töötaks kindlalt 100% juhtude puhul. Sellised faktorid, nagu ostutšekkipildi kvaliteet ning otsutšekki seis oluliselt mõjutavad lõplikku tulemust, ja see on võimatu täielikult eemaldada nende faktorite mõju kasutades isegi keerulisi pilditöötlusmehhanisme. Seega, et teha kulu informatsiooni ostutšekki pealt väljalugemise funktsionaalsust veel resultatiivsemaks, järgmisena võiks kaaluda masinõppe mehhanismi implementeerimist, mida saaks trennida nende ostutšekkide peal, millest suudeti edukalt kulu informatsiooni välja lugeda.

# References

[1]

[Swift for iOS [https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift\_Programming\_Language/]]

[Native app definiton https://msdn.microsoft.com/en-us/library/jj149679.aspx]

<http://cdn.intechopen.com/pdfs-wm/11405.pdf>

<http://static.googleusercontent.com/media/research.google.com/en//pubs/archive/33418.pdf>

<http://www.ipol.im/pub/art/2011/bcm_nlm/>

<http://www.periodicooficial.oaxaca.gob.mx/files/2011/05/EXT02-2011-05-19.pdf> [Concurrency in practice]

<http://www.json.org/xml.html> [JSON vs xml]

<http://www.slideshare.net/jmusser/open-apis-state-of-the-market-2011> [JSON popularity]