# 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 receipt in order to automatically recognize and extract basic expense information from the 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 applciation, focusing on possibility of expense information recognition from Estonian receipts as well as usability and security aspects of the applicaiton.

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.

The knowledge gained during development of this applciation 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
* Reliability evaluation of the application’s function of expense information recognition from the aspect of Tesseract for OCR and OpenCV for image preprocessing usage.

## Goals

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

1. Possibility to take picture of a receipt, recognize total sum of the purchase and the company the purchase was from based on the receipt image, compose an expense entry from this information an 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.
5. Graphics? (to be implemented)

## 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 serice of the e-business register. Both service and client are also using third-party libraries.

## Thesis overview

//to do: add when is ready.

# Theory

## Image preprocessing before optical charecter recogniton

In the author’s mobile application, the picture of receipt used for further exepnse 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 charecter recogniton. Besides that, the recognition stage is made even harder by the fact that cash registers use mainly either thermal printers with thremal 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 charecters are composed of small dots, which makes it hard for the OCR engine to determine them correctly. As a result, in order to enchance 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 enchance 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 tend 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 minum 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 threshholding 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 charecter 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 exact features are 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. //todo add

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

# System overview

## 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 its is to possibile 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 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 the 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 authors estimations, up to 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 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 favour of descreasing 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 soilution:

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 applciations for multiple platforms, then expese 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. Furhtermore, 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 the 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 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, behaviour, 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 possibilility 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 prerspective 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 applicaiton

A web mobile application is another widespread mobile applciation type. Such kind of application is generally a web-page, which is accessible through the web browser of the device. Such kind of applciation 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 to invoke device’s camera. Mobile web applications also have poor support for working offline.

#### Hybrid mobile application

Hybrid movile applicaiton 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. Consecuently, 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 applicaiton. The pure web approach did not meet the needs of the author, as the application beign 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 applciation 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, better performance of the native approach

### Images processing

#### Initial synchronous solution

In the beginning, when the author designed the initial solution for image processing, 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, receiveng 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 necessarry to determine, which stage of the process was the so called “bottleneck”. After some investigation it turned out, that the request processing timeline propogated 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 10-11 seconds. This meant, after a user uploaded a receipt image, the response considering succesful 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, consecuently¸ 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 queue, which could be of FIFO type and a dispacther 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, but in such case one should remember, that in case of computation intensive tasks, there will be no use of too many processing threads - the best result will be achieved with N + 1 threads, where N is the number of available CPU-s. 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 organised 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 necessarry 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 favour 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 notificaiton is that it does not require the application to be running in order to reach the target device, so the user, who have uploaded the receipt image will receive it even if he closed the application after uploading the image. The device will receiver the 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 interchaning the data between the client and the server side, as it is lightweigh 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

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 somewhere, 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.

[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]