TALLINN UNIVERSITY OF TECHNOLOGY
Faculty of Information Technology
Department of Computer Science
Chair of Network Software

Expense tracking mobile application with receipt scanning functionality

Bachelor's thesis

Student: Roman Kaskman
Student code: 113089 IAPB
Advisor: Roger Kerse

Tallinn
2015

Author's declaration

I declare that this thesis is the result of my own research except as cited in the references. The thesis has not been accepted for any degree and is not concurrently submitted in candidature of any other degree.

25.05.2015 Roman Kaskman

(date) *(signature)*

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 extracted expenses information 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 expense 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 expenses information extraction from receipt images passes the testing phase, conclusion regarding its reliability is made. Moreover, proposals for further improvements of the application's functionality are also presented.

The thesis is in English and contains 53 pages of text, 6 chapters and 14 figures.

Annotatsioon

Käesoleva bakalaureusetöö eesmärk on luua mobiilirakendus kasutaja kulude üle arvestuse pidamiseks ja dokumenteerimiseks. Lahenduse peamine fookus on funktsionaalsusel, mis võimaldab teha pilte Eesti ettevõtete väljastatud ostutšekkidest, lugeda ostutšeki pildi pealt välja kuluinfo ning salvestada see autenditud kasutaja kulude nimekirja.

Töös käsitletavad põhilised probleemid on sobiva rakenduse arhitektuuri- ja disainilahenduse väljatöötamine rakenduse jõudluse, kasutatavuse, turvalisuse ja edasiarenduse seisukohtast ning ostutšeki pildi pealt kulu info väljalugemise efektiivsete võimaluste uurimine ning teostamine.

Töö tulemusena valmib mobiilirakendus kulude üle arvestuse pidamiseks Androidi platvormile. Pärast ostutšekkidelt kulude väljalugemise funktsionaalsuse testimisfaasi tehakse järeldusi selle toimimise ja töökindluse kohta. Lisaks tuuakse välja ka võimalikke edasiarendusvõimalusi ja parandusettepanekuid valminud mobiilirakenuse jaoks.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 53 leheküljel, 6 peatükki ja 14 joonist.

Glossary of terms and abbreviations

API	Optical character recognition
APK	Android application package
Base64	Binary data text representation encoding
CPU	Central processing unit
CRUD	Create, read, update and delete
FIFO	First In, First Out
Google Developers Console	Google environment for binding Android applications 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 character recognition
ORM	Object-relational mapping
POJO	Plain old java object

REST	Representational State Transfer
Scraper	A program assembling content from web pages
SDK	Software development kit
SQL	Structured Query Language
SSL	Secure Sockets Layer
URI	Uniform resource identifier
XML	Extensible Markup Language

Table of figures

Table of contents

1. 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 (e.g. 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 images. Hence, a free application of such kind based on open-source technologies can be beneficial for users not willing to pay for the proprietary solutions. 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.

1.1 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 appeared 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, for example, be integrated with some existing 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 text from a receipt.
- Complexities and constraints related to development of application of this type.

1.2 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 expense recognition from a receipt image 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.

1.3 Methodology

To achieve the goal the author implemented Android [3] client-side application and server-side application written in Java programming language [4]. Both client and server are designed in an object-oriented manner. The server-side is put to communicate with the database (PostgreSQL) [5] and an external web service of the Estonian e-business register [6]. Both service and client are using third-party libraries.

1.4 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 receipt images.

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 the context of this application.

The fourth part of the thesis introduces a brief overview of the chosen technologies. It is also explained, for which purpose 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 implemented application, analyses its reliability as well as proposes possible further improvements, which should be considered in the future development.

2. Theory

2.1 Image preprocessing before optical character recognition

In the author's mobile application, pictures of receipts used for further expense information extraction are taken using mobile device's camera. As not all the mobile devices have high resolution cameras, this means that the quality of captured receipt images 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 [7] with thermal paper or dot matrix printers [8] for receipt printing. The problem of the receipts printed with the thermal printer is that they tend to fade in case of even minor abrasion, 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.

2.1.1 Conversion to grayscale

The first measure to be taken in order to enhance image for further OCR is converting color image to grayscale image. Grayscale image is an image in which value of each pixel carries only intensity information. Images of this sort, also known as black-and-white, are typically composed of 256 shades of gray, varying from black at the weakest intensity to white at the strongest [9]. Conversion of the image to grayscale is important from the perspective of making further image processing stages faster [10].

2.1.2 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 takes place during the acquisition 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 [11]. 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. The filter performing such kind of operation is called non-local means filter [12].

2.1.3 Morphological closing

As it was mentioned above, the level of quality of receipt print is frequently quite low. 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 purpose of linking the parts of a symbol separated with the gaps, an operation of morphological closing should be applied [13].

This operation is comprised of two basic morphological operations: dilation, followed by erosion [14]. 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 (see Figure 1).

Figure 1. A symbol before and after application of morphological closing.

2.1.4 Thresholding

The operation of thresholding is considered to be a simple method of image segmentation. It takes a grayscale image (see Figure 2) as an input and transforms it into a binary image (see Figure 3), 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) [15].

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 to 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 surrounding pixels of each pixel in the specified area of the image [16].

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 an input for recognition by the OCR engine.

Figure 2. Receipt image before thresholding.

Figure 3. Receipt image after thresholding. The color scheme is inverted.

2.2 Optical character recognition

Optical character recognition, or simply OCR, ismachine 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.

There are various approaches used for OCR engines design:

- Matrix matching: Each character 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 [18].
- 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 [18].
- Neural Networks: The main idea of this approach is to train an artificial neural network with a set of training data input, which would be used for the further recognition [18].

3. System design considerations

3.1 Choice of the architecture

Before starting 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:

   a) No need to implement the server-side and thus it is 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 accessible from device's local storage.
   b) The mobile application does not need Internet connection, as all its functionality would be available offline.

   Cons of the solution:

   a) 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 keep track on all of his expenses on multiple devices simultaneously, user would have to manually duplicate the expense entries, which may be very inconvenient.
   b) 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 the user's expenses data would get lost. Such approach makes the application unreliable from the user's perspective.
   c) 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. This will considerably increase the size of the installed application (according to the author's estimations, the

size can be more than 60 megabytes). This fact can prevent users from downloading and installing it.

d) If there is a plan to introduce support for the other mobile platforms besides Android (e.g. iOS [19], Windows Phone [20]) in the future, there may occur compatibility issues, 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.

2. 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 stage 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 expenses information.

   Pros of the solution:

   a) User may have a single account and use it with numerous devices, and all the expenses data would be automatically synchronized with the server. There is no threat of losing the expenses data as in case of the previous solution, because all the data will be stored in the database on the server-side.
   b) 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 of the posted 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.

3. Develop a mobile application (client) and a server. The mobile client should be thin and all the business logic should be located on the server-side. The expenses 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:

a) Same as the point a) of the 2nd solution
b) The client-side is thinner, as it does not include libraries for image preprocessing and OCR. As a result, the application size is considerably smaller (less than 20 megabytes), making it more attractive for the user.
c) The whole application is more consistent - if there are client applications for multiple platforms, then expenses information extraction from receipts works in the same way for all the devices.
d) Expenses information extraction from receipt images techniques may require continuous 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 every platform. 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:

a) 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 solutions, 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 has a few arguable pros and significant cons (particularly cons point a) and b)), which contradict such aspects of application as reliability and usability.

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 its pros points b) and d).

3.2 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 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, decisions regarding the system structure, behavior, data exchange protocols, data exchange formats as well as data storage options should be made.

3.2.1 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.

3.2.1.1 Native application

A native mobile application is written using device's platform-specific language (for instance Swift [21] for iOS, Java [22] for Android), targeting the API and features of the selected platform with the use of 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 [23]. Furthermore, native applications tend to have better performance in comparison with web-based solutions [24].

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 work on a large variety of devices, one must know and consider all the possibilities and constraints of the API-s as well as hardware limitations of particular devices.

3.2.1.2 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 a mobile device. Web mobile applications are created using HTML, CSS and JavaScript. As any other

web content, they are retrieved from the server using Internet. At the same time, mobile web application's design must be adapted for the size of the target devices [23].

There are numerous benefits of developing a mobile application as a web application. Firstly, familiar web technologies are used for that, so it does not require knowledge of platform-specific domain, 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 with a new request.

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 of web browsers. Furthermore, web applications have limited access to some device's platform-specific or hardware features. Though, for example HTML5 allows invoking of the device's camera [25], it is impossible to make a web application on Android process incoming push notifications or access device's notification manager. Mobile web applications also have relatively poor support of working offline [23].

3.2.1.3 Hybrid mobile application

A 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 to show application web views. At the same time, a hybrid has a possibility to use all native app features [23]. The described effect is achieved with the use of special frameworks, such as Apache Cordova [26]. 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 native look and feel for each platform because of the shared web view as well as lack of responsiveness in comparison with the native applications.

3.2.1.4 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 functionality may also be a problem in case the web approach is chosen.

Releasing the application for multiple platforms with least effort was not among the goals of 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 native applications.

3.2.2 Images processing flow

3.2.2.1 Initial synchronous solution

In the beginning, when the author designed the initial solution for receipt images processing (images preprocessing, OCR, expense information extraction), it was fully synchronous. Synchronous means that a receipt image was posted from the client-side to the server with a HTTP request, and the client-side had to wait for the response with the extracted receipt data until it arrived. On the client-side 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 response was handled as soon as it arrived.

Though from the client-side perspective everything seemed to work correctly, such kind of approach was unsustainable, mainly because the total time between sending the request and receiving the response was ~15 seconds, and it was too long from the perspective of the server throughput. As a server has a finite number of worker threads (for example, for Tomcat 8.0 server the default number is 200) [27] 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 these requests will have to wait for some worker thread to become free for processing, and this may result in receiving request timeouts. From the perspective of the user experience, receiving timeouts is not a good practice, so the author considered finding out better solution instead of the described synchronous approach.

3.2.2.2 Asynchronous solution

To find out the best approach for designing an alternative processing solution, it was first necessary to determine, which stage of the process was 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 has uploaded a receipt image, the response considering successfully uploaded image (or an error response in case of an erroneous scenario) could be sent to him, while 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 (see Figure 4). Such approach can improve server throughput and increase number of idle worker threads ready to process arriving requests and, consequently, help to avoid possible 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 retrieve awaiting images for further 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 find the pending images in the right order.

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

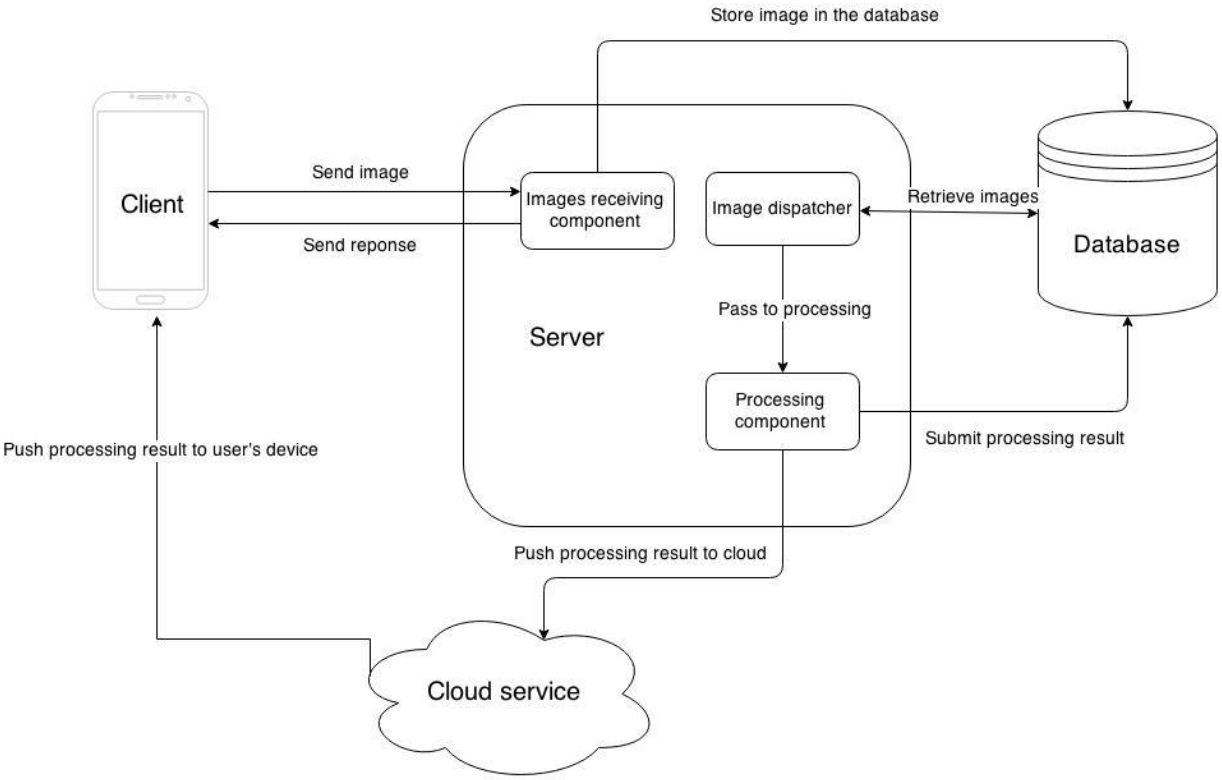

Figure 4. Diagram representing asynchronous image processing flow.

3.2.2.3 Notifying the user about image processing result

As receipt image processing is organized in an asynchronous way, the client-side cannot exactly determine whether processing of an image has finished 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 requests. 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 optimal 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 on the server. 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 [28] for Android should be used. A valuable 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 left the application after uploading the image. The device will receive a wake-up event as soon as the notification arrives, a notification event will be passed to a corresponding service, and the user will be able to see extracted expense information, or an error in case recognition was not successful.

3.2.3 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 [29]. What is more, in recent years JSON has become a standard for the newer generation of web applications [30].

3.2.4 Authentication and authorization

To use the functionality of the application, the user must be authenticated and have corresponding user rights. When a user passes the authentication stage, a session for him should be created. Logged-in user data (user name, e-mail) should be then stored locally on the user's device. The authentication mechanism should be also designed in a way, which allows its further extension by adding new means of authentication.

3.2.5 Offline data storage

If the user has logged in once, later he has an opportunity to capture images of the receipts while being offline. When the Internet connection becomes available, he will be able to send the images to the server. The captured images must be stored on a device, and a suitable place for storing them is an on-device database, as it will make it easy to retrieve the images in case the device is used by multiple users and every user wants to see only his own pending images.

4. Technologies

This paragraph describes technologies, which were used for the development of both server and client part of the application. In each subparagraph there is a brief description of corresponding technology and a short remark saying where the technology was applied in the application. More precise explanations of how the described technologies were used can be found in the 5th paragraph (Implementation).

4.1 Choice of platform and programming language

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

The best candidates for implementing the server were Java, Ruby [32] and Python [33], as these are considered to be the most mature platforms for designing a REST service [34]. 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 [35] [36].

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# [37], 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.

4.2 Spring framework

Spring is a Java-based development framework created with the purpose of enhancing 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 [38].

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 loosely coupled and reusable components, which become the building blocks for the application.

- Design of the REST service controller layer, which maps URI resources to corresponding methods on the server-side.

In the described project the author uses Spring Boot – a sub-project of the Spring framework. The main advantage 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 of running an application as a *jar* file on an embedded server (Tomcat 8.0 in this project) instead of deploying a *war* file in the container [39].

4.3 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 [40].

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

4.4 Hibernate ORM

Hibernate ORM is an object-relational mapping solution, created for Java programming language. 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 [41].

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, or a set of joined tables, is done either in XML configuration file or with the use of Java annotations directly in the class being mapped. Such approach prevents a developer from writing extra SQL queries as well as boilerplate code in order to map results of the database queries to Java objects.

In this project Hibernate ORM is used on the server-side for performing CRUD operations. It is also worth pointing out that the author does not use Hibernate ORM directly, but rather uses Java Persistence API (JPA), and in this case Hibernate ORM served as an implementation of the JPA specification. Such approach favors programming using interface rather than implementation and therefore makes 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.

4.5 PostgreSQL

PostgreSQL is one of the most powerful object-relational database systems, which is fully open source [8]. 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 are used. Hibernate ORM uses Java Database Connectivity (JDBC) to perform the database reading and writing operations.

4.6 Jackson

Jackson is a Java library providing a set of data-processing tools with the main focus on JSON parsing and generation, as well as binding JSON with Java POJOs [42].

In the current project Jackson library is applied 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 JSON-format response body from Java objects.

4.7 Dagger

Dagger is a library used for 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 [43]. As a result, an application is built of interchangeable, reusable and loosely coupled components.

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

4.8 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 [44].

In the current project, Retrofit is used on the client-side of the application for communicating with the server via HTTP.

4.9 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. [45].

The library itself is written in C++ programming language, but it also has a Java wrapper which uses Java Native Access (JNA) for purpose of invoking the native code [45].

The library has a set of comprehensive tools for image processing, which allows to perform operations of changing color spaces, geometric image transformations, thresholding operations as well as morphological transformations. Mainly because of this functionality this library is a suitable candidate for performing image preprocessing on the server-side of the application.

4.10 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 [46].

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 as a sample 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 [47].

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 [47].

Since version 2.00 Tesseract OCR is fully UTF-8 encoding capable [48]. Tesseract OCR is written in C++, but a Java wrapper called Tess4j [49] can be used for Java projects.

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

5. Implementation

This paragraph covers the implementation process of the application, explaining in details the most important stages of this process and explaining usage of the above mentioned technologies and design solutions. The client and the server-side development process is covered together, mainly because these parts were incrementally developed in parallel.

The links to the source code of the application can be found in Appendix.

5.1 Authentication

For the purpose of authentication Google Play Services [50] were used. It is a convenient choice from the user's perspective, as it does not require any additional account registration, 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 [51] protocol, which allows granting a third-party limited access to the protected resources of the user without a need to pass user credentials to this third-party.

5.1.1 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 [63] 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 the registration the application can make calls to the Google Play Services API. An Android application must also include Google Play Service library in order to communicate with Google Play Service.

5.1.2 Client-side token retrieval flow

When the Android application starts, 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 user is his Google user id and user name.

The access token is retrieved within a synchronous transaction over the Internet, that is why this transactions must be executed by a separate worker thread to prevent blocking of application's user interface. For this purpose special Android asynchronous task was implemented.

5.1.3 Token transmission security prerequisites

After the access token arrives to the client-side, it must be sent to the server for further validation. If the token proves to be valid, then a session for the user is initiated. The access token is transmitted from the client-side to the server over HTTPS, so that communication between the client and the server is secure. In order to prepare the server to accept HTTPS connections, an SSL certificate is required.

As the current goal of the thesis 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. Instead, a self-signed certificate was created for testing purposes. In order to generate a certificate to be used by the server, a command-line utility named Java Keytool was used. The certificate was generated with the use of RSA [52] algorithm for creating the key pair, each key with the size of 2048 bits. PKCS #12 [53] was used as the key store type of the certificate.

In order the Android application could check that the self-signed certificate of the server is a trusted one, a custom certificate trust manager implementation (see Figure 5) 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 expected Common Name (CN) and its public key is equal to the public key of a trusted certificate, then this certificate proves to be trusted.

```
public class CertificateTrustManager implements X509TrustManager {
    private Map<String, X509Certificate> trustedCerts = new HashMap<>();

    @Override
    public X509Certificate[] getAcceptedIssuers() {
        return new X509Certificate[0];
    }

    private void checkTrusted(X509Certificate[] chain) throws CertificateException {
        if (chain.length != 1) {
            throw new CertificateException("Invalid cert chain length");
        }
        X509Certificate trustedCert = trustedCerts.get(
                getCnFor(chain[0]));
        if (trustedCert == null) {
            throw new CertificateException("Untrusted certificate");
        }
        if (!Arrays.equals(chain[0].getPublicKey().getEncoded(),
                trustedCert.getPublicKey().getEncoded())) {
            throw new CertificateException("Invalid certificate");
        }
        trustedCert.checkValidity();
    }

    @Override
    public void checkClientTrusted(X509Certificate[] chain, String authType)
            throws CertificateException {
        checkTrusted(chain);
    }

    @Override
    public void checkServerTrusted(X509Certificate[] chain, String authType)
            throws CertificateException {
        checkTrusted(chain);
    }
```

Figure 5. Custom certificate trust manager implementation.

5.1.4 Server-side authentication

The author has chosen Spring framework for implementing the server-side of the application. Spring framework allows programming custom filter objects, which can intercept and filter requests before they reach the target resource of the server.

A custom implementation of an authentication filter (see Figure 6) 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 the request is made. If no token is found from the request, then an exception is thrown. If the access token is successfully retrieved from the request, then authentication with the provided token is initiated.

Authentication is executed by Spring Security module component. This component needs an implementation of the authentication-providing class as an input. For this purpose the author designed a custom implementation of an authentication provider, which uses OAuth 2.0 access tokens as a means of authentication. First of all, this authentication provider makes a call to the Google API in order to retrieve the information about the provided access token in JSON format. If such token is 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 the expected one, then Google user id and user name are obtained by the server from Google API, otherwise the authentication process is interrupted. In the end, the session for the user is initiated. Google user id and user name then become accessible on the server-side from an object representing an authenticated user (see Figure 7). Session expiry time is transferred to the client-side in HTTP response headers.

```
public class AuthenticationFilter extends GenericFilterBean {
    private static final String AUTH_TOKEN_PARAM = "authToken";
    private AuthenticationManager authenticationManager;

    public AuthenticationFilter(AuthenticationManager authenticationManager) {
        this.authenticationManager = authenticationManager;
    }

    @Override
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain filterChain)
            throws IOException, ServletException {
        SecurityContext securityContext = SecurityContextHolder.getContext();
        Authentication authentication = securityContext.getAuthentication();

        if (authentication == null) {
            authenticate (request);
        }
        HttpServletResponse httpResponse = cast(response);
        HttpServletRequest httpRequest = cast(request);

        long maxInactiveIntervalInMs = httpRequest.getSession().getMaxInactiveInterval() * 1000;

        httpResponse.setHeader("expiresAt", String.valueOf(maxInactiveIntervalInMs));
        filterChain.doFilter(request, response);
    }
```

Figure 6. Authentication filter implementation code.

```
public static User getAuthenticatedUser() {
    return (User) SecurityContextHolder.getContext().getAuthentication().getPrincipal();
}
```

Figure 7. Accessing authenticated user information.

5.1.5 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. The expiry time is updated with each response received from the server. Client activities, which perform calls to the server, or authentication-aware activities, check whether the session has expired and initiate authentication in case of expiry. In case of successful authentication, the application usage flow which might have been interrupted by authentication, continues.

5.2 Receipt image processing and OCR

A user captures the image of the receipt with the use of device's camera. In case the image is successfully is captured and there is Internet connection available, the image is sent to the server for further processing. The image is encoded with Base64 encoding in order to be sent in JSON string representation. When the image reaches the server, it is placed into a processing queue, represented by a database table mapped to a Java class using JPA annotations (see Figure 8). A HTTP response is sent to the user as soon as the image is saved in the database.

```
@Entity
@Table(name = "receipt_picture_data")
public class ReceiptImageWrapper {
    public static final String STATE_NON_PROCESSED = "N";

    @Id
    @SequenceGenerator(name = "receipt_picture_data_seq", sequenceName = "receipt_picture_data_seq")
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "receipt_picture_data_seq")
    @Column(name = "id")
    private Long id;
    @Column(name = "user_id")
    private String userId;
    @Column(name = "registration_id")
    private String registrationId;
    @Column(name = "regnumber_picture")
    @Lob
    private String regNumberPicture;
    @Column(name = "reg_number_picture_extension")
```

Figure 8. Receipt image entity mapped to a database table.

As it was previously mentioned in the paragraph 3.2.2, image processing (preprocessing and OCR) is designed to be executed asynchronously. Asynchronous processing is performed as soon as there is free resource to do that, i.e. there are free worker threads available. A dispatcher thread retrieves a batch of images for processing, retrieving the oldest images first. These images are then passed to a number of threads, which process them in parallel. The number of threads in the thread pool to perform processing is N + 1, where N is the number of CPU-s on the machine. Because image processing is a compute-intensive task, such number of threads was chosen for the system to achieve its optimum utilization in this particular case [54]. When there are no more images to process in the database, the dispatcher thread is put to sleep for 5 seconds before it repeats the request for new images.

As Hibernate ORM is used for CRUD operations in the databases, images are retrieved from the database using HQL (Hibernate Query Language) [55], which resembles SQL to some extent, but the syntax of its queries is based on the level of Java objects corresponding to the database tables.

During the processing phase the image is first decoded from Base64 string and written into a file. Then image preprocessing then takes place. During the preprocessing phase operations of conversion to grayscale, denoising, morphological closing and thresholding are applied to the image using corresponding methods provided by OpenCV. As it was already mentioned in the paragraph 2.1, these operations are done in order to get a better recognition probability by the OCR engine.

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 an OCR-executing object (each image-processing thread uses a separate object), setting the output language (the combination of Estonian and English proved to work in the best way) and invoking the OCR-executing method, which takes the preprocessed image as an input argument (see Figure 9).

```
private String performOCR(File imageFile) throws TesseractException {
    Tesseract1 tesseract = new Tesseract1();
    tesseract.setLanguage("eng+est");
    return tesseract.doOCR(imageFile);
}
```

Figure 9. Performing OCR with Tesseract OCR.

5.3 Receipt information extraction from OCR result

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

If one takes a look at 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 is organized and represented on receipts – each enterprise does it the own way. This makes extraction of the required information complicated, as the solution responsible for information extraction must consider different formats of information representation.

5.3.1 Enterprise name extraction

When it comes to location of the enterprise name of the receipt, then usually it can be found in the "header" part of the receipt. However, there are no any exact rules that name of the enterprise is, for example, always located on the first line on the receipt or in some other distinct area. Consequently, 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 [56]. 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. Unfortunately, the service is not that simply accessible – it requires signing an agreement for using the service as well as paying for its usage [57] [58]. 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.

5.3.2 Total sum extraction

The total sum entry of a receipt is usually located in its bottom part. The text extracted from an image by the OCR has the original line endings, so it is possible to determine where a line starts or ends. For total sum extraction the following logic is 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.

5.3.3 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 not to 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) [59] library was used. FREJ allows to create regular expressions (the syntax is 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 them to find the required keywords. It is possible to define how strict the matching should be by 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 the provided pattern 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.

5.4 Notifying the user about a processed expense

In case both enterprise name and total sum were extracted, the extraction is considered to be successful, otherwise it is erroneous. If extraction is successful, an extracted expense is persisted in the database in an initial state, i.e. it still needs 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.

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

5.4.1 Client-side notifications handling

In order to integrate Google Cloud Messaging into the client part of the application, Google Play Services library must be present among the dependencies of the Android application. An application must be also registered in the Google Developers Console [60]. 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 of the device 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 wakeful broadcast receiver 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 notification manager. The user can see the arrived notifications in the notification area (see Figure 10) and press them in order to see information concerning the result. In case of a successful result (see Figure 11), a confirmation activity for an expense, allowing 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 (see Figure 12).

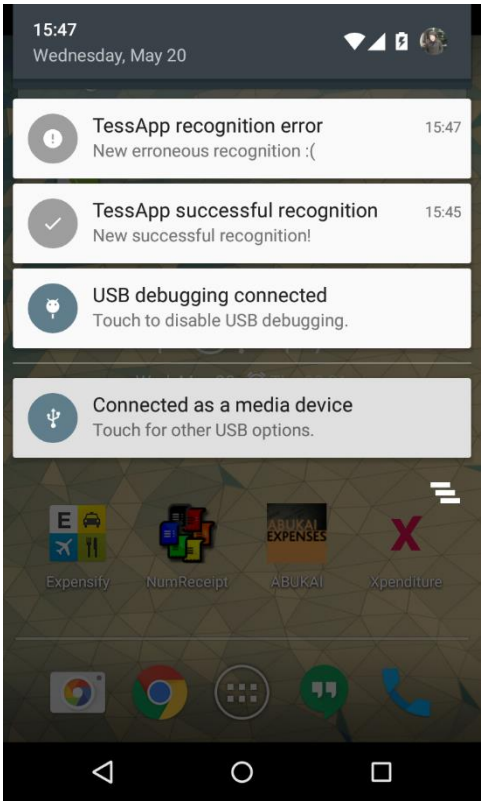

Figure 10. Processing result notifications

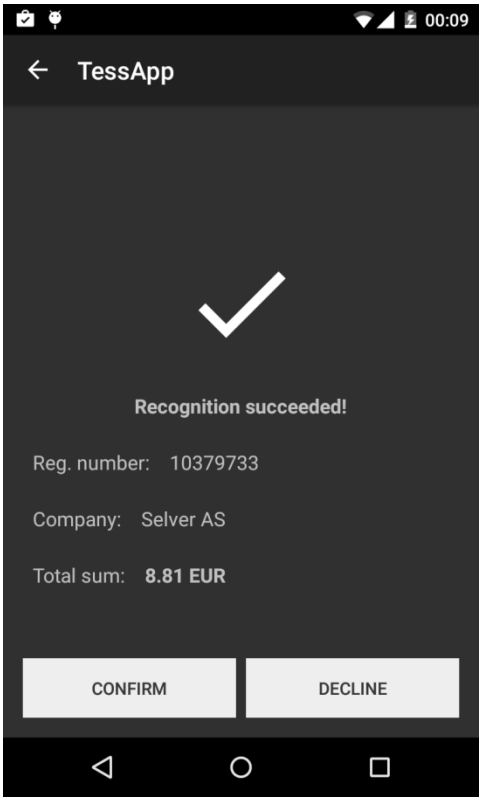

Figure 11. Successful recognition notification.

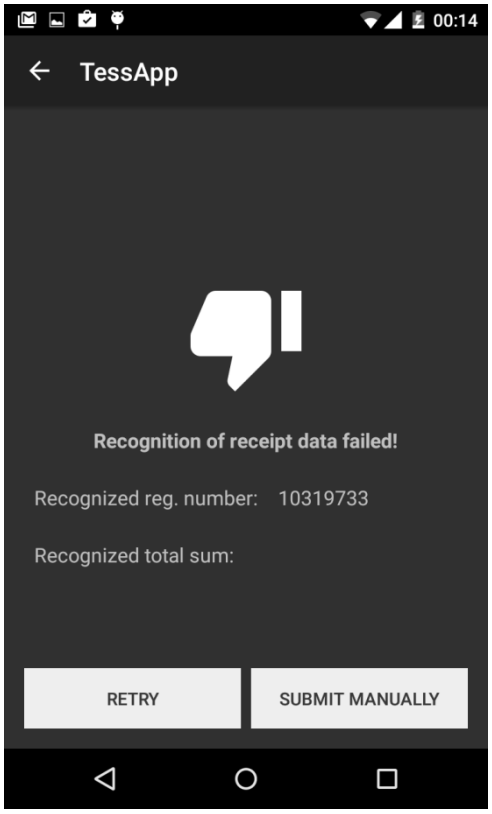

Figure 12. An example of erroneous recognition. In this case total sum was not recognized.

5.4.2 Server-side notifications handling

The server sends 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, 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.

5.5 Displaying user's 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 the user 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 (see Figure 13).

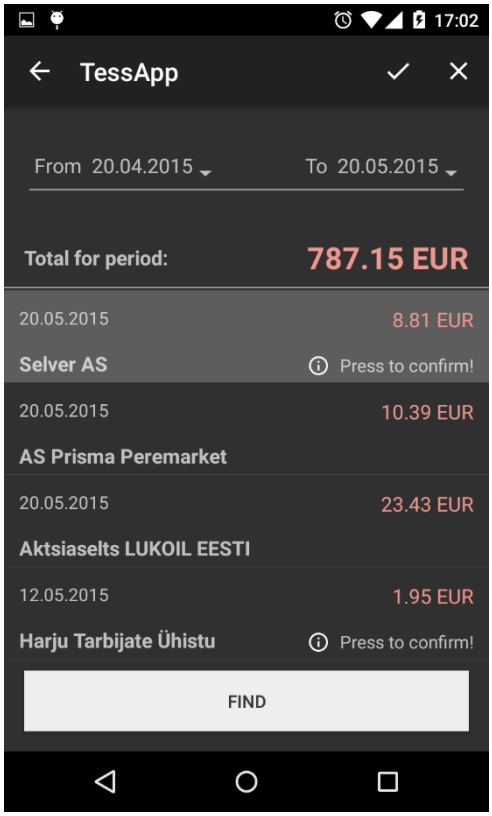

Figure 13. Expense list provides a possibility to confirm or decline recognized expenses.

5.6 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 in a file system, and its location in the file system is stored in SQLite database [61], which is provided by Android SDK. This makes it convenient 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, allowing either to send the selected image for further expense extraction or just to delete it. If a pending image is successfully sent to the server for expense extraction, then both reference in the database and the image itself are deleted.

5.7 Expense manual submission

The application has an opportunity to submit an expense manually in case extraction of expense information from the receipt is failing due to poor quality of the 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 case is retrieved using the enterprise registration number in the same way as in case of recognition of expense from a receipt.

5.8 Application dashboard

The application dashboard was implemented to be the main view of the client-side application and serve as an access point to all the other functionality. On the dashboard it also possible to see the user logged in into the application (see Figure 14).

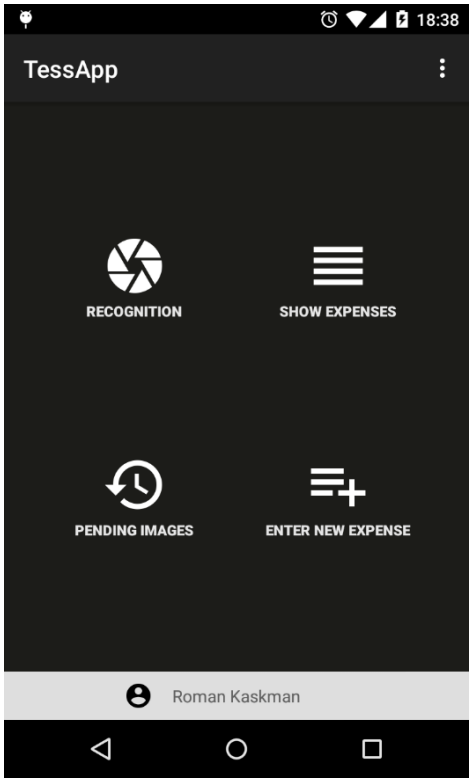

Figure 14. Application dashboard

6. Implemented application

6.1 Testing expense extraction from receipt functionality

6.1.1 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 recognitions. It proved that preprocessing stage and fuzzy matching were useful enhancements for improving recognition rate.

6.1.2 Analysis of the testing result

The recognition rate of 6.2 out of 10 may be considered quite successful 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, 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 a different font, which appears only one or two times on a receipt. Secondly, recognition was invalid in some cases because 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.

6.2 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 [62]) 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 communication 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 performs 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.

6.3 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 must 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, an Android mobile application 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 further simple integration for other mobile platforms besides Android, as they will be using the same REST API provided by the 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 architecture 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 of further improvements. Though from the perspective of expense extraction from a receipt image the application has shown 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 significantly, and it is impossible to completely eliminate the influence of these factors solely by using even advanced image processing mechanisms. That is why in order to improve the recognition rate introducing a machine learning mechanism, which would be trained with successfully recognized receipts, could be the next goal for the future.

Kokkuvõte

Töö eesmärk on luua mobiilirakendus, mis on mõeldud kasutaja kulude üle arve pidamiseks ja dokumenteerimiseks. Mobiilirakendus peaks võimaldama teha pilte Eesti ettevõtete poolt väljastatud ostutšekkidest, ostutšekkide pealt automaatselt leida informatsioon kulu kohta ja lisada saadud informatisoon kasutaja kulude listi. Lisaks sellele peaks rakendus võimaldama kasutaja turvalist autentimist ning pakkuma võimalust näha kasutaja kulusid valitud ajaperioodil.

Töö tulemusena valmis mobiilirakendus Androidi platvormi jaoks, mis vastab kõigile ülal toodud nõuetele. Kogu rakenduse funktsionaalsus, sealhulgas ostutšekkide piltide eeltöötlemine, optiline tähtede tuvastamine eeltöödeldud ostutšeki pildi pealt ning tuvastatud tulemusest vajaliku kulu informatsiooni leidmine, oli realiseeritud kasutades avatud lähtekoodiga tehnoloogiaid. Seega ei pea taolise rakenduse loomiseks ilmtingimata kasutama tasulisi litsenseeritud tarkvarakomponente. Süsteemi disain võimaldab luua klientrakendusi lisaks Androidile ka teiste mobiiliplatvormide jaoks, sest loodavad klientrakendused saavad kasutada sama serveri REST API-d.

Töö autor analüüsis võimalikke viise rakenduse arhitektuuri organiseerimiseks ning kirjeldas lisaks ka erinevate tarkvarakomponentide disaini ning nende komponentide koostoimet. Töös toodi välja põhilised poolt argumendid klient-server arhitektuuri valimiseks, *native*-rakenduse arenduse ning asünkroonse ostutšeki piltide töötlemiseks ning ühtlasi võrreldi neid lähenemisi võimalike alternatiividega.

Saab väita, et töö eesmärk on saavutatud, kuna valminud rakendus töötab nii, nagu oli eeldatud, olgugi et on võimalik välja tuua parandusettepanekuid, mida tulevikus realiseerida. Kuigi kuluinfo ostutšeki pealt väljalugemise funktsionaalsus toimib suhteliselt hästi, on väga keeruline teostada seda nii, et see töötaks kindlalt 100% juhtude puhul. Sellised näitajad, nagu pildi kvaliteet ning otsutšeki seisukord mõjutavad oluliselt lõplikku tulemust ja on võimatu täielikult kõrvaldada nende mõju isegi keerulisi pilditöötlusmehhanisme kasutades. Selleks, et teha kulu informatsiooni ostutšeki pealt väljalugemise funktsionaalsust võimekamaks, võiks kaaluda masinõppe mehhanismi teostamist. Loodud mehhanismi saaks treenida nende ostutšekkide järgi, millest suudeti edukalt kuluinfo välja lugeda.

References

[1] Expensify mobile application homepage [WWW] http://use.expensify.com/ (14.05.2015)

[2] Xpenditure mobile application homepage [WWW] https://xpenditure.com/en/ (15.05.2015)

[3] Android homepage [WWW] http://www.android.com/ (15.05.2015)

[4] Java Platform Standard Edition 7 Documentation [WWW] http://docs.oracle.com/javase/7/docs/ (15.05.2015)

[5] PostgreSQL homepage [WWW] http://www.postgresql.org/ (15.05.2015)

[6] Estonian e-business register [WWW] https://ariregister.rik.ee/ (15.05.2015)

[7] Thermal printer definition [WWW] http://www.pcmag.com/encyclopedia/term/41434/direct-thermal-printer (15.05.2015)

[8] Dot matrix printer definition [WWW] http://www.pcmag.com/encyclopedia/term/41904/dot-matrix-printer (15.05.2015)

[9] Grayscale definition [WWW] http://www.lib.umich.edu/files/short_DCU_Terms_Definitions.pdf (15.05.2015)

[10] Patel, C., Patel, A., Patel, D. (2012). Optical Character Recognition by Open Source OCR Tool Tesseract: A Case Study. – *International Journal of Computer Applications (0975 – 8887) Volume 55– No.10*, 52-53

[11] Buades, A., Coll, B., Morel, J.M. (2006). A review of image denoising methods, with a new one. – *Multiscale Modeling and Simulation, Vol. 4 (2)*, 490-530

[12] Buades, A., Coll, B., Morel, J.M. (2005). A non local algorithm for image denoising. – *IEEE Computer Vision and Pattern Recognition 2005, Vol 2*, 60-65

[13] OpenCV morphological transformation tutorial [WWW] http://docs.opencv.org/doc/tutorials/imgproc/opening_closing_hats/opening_closing_hats.html (16.05.2015)

[14] OpenCV morphological erosion and dilatation tutorial [WWW] http://docs.opencv.org/doc/tutorials/imgproc/erosion_dilatation/erosion_dilatation.html (16.05.2015)

[15] Fisher, R., Perkins S., Walker A., Wolfart, E. (2003). Thresholding [WWW] http://homepages.inf.ed.ac.uk/rbf/HIPR2/threshld.htm (16.05.2015)

[16]] Fisher, R., Perkins S., Walker A., Wolfart, E. (2003). Adaptive thresholding [WWW] http://homepages.inf.ed.ac.uk/rbf/HIPR2/adpthrsh.htm (16.05.2015)

[17] OCR definition [WWW] http://www.pcmag.com/encyclopedia/term/48267/ocr (16.05.2015)

[18] Singh, S. (2013) Optical Character Recognition Techniques: A Survey. – *Journal of Emerging Trends in Computing and Information Sciences, Vol. 4, No. 6*, 545-546

[19] iOS operating system [WWW] https://www.apple.com/ios/ (16.05.2015)

[20] Windows phone platform [WWW] https://www.windowsphone.com/en-us (16.05.2015)

[21] Swift programming language [WWW] https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/ (16.05.2015)

[22] Android SDK requirements [WWW] https://developer.android.com/sdk/index.html#Requirements (16.05.2015)

[23] Choosing between web and native experience [WWW] https://msdn.microsoft.com/en-us/library/jj149679.aspx (16.05.2015)

[24] Jobe, W. (2013). Native Apps Vs. Mobile Web Apps. – *International Journal of Interactive Mobile Technologies, Vol 7, No 4*, 28

[25] HTML media capture [WWW] http://www.w3.org/TR/html-media-capture/ (17.05.2015)

[26] Apache Cordova framework [WWW] https://cordova.apache.org/ (17.05.2015)

[27] Apache Tomcat 8.0 documentation [WWW] https://tomcat.apache.org/tomcat-8.0-doc/config/executor.html (17.05.2015)

[28] Google Cloud Messaging [WWW] https://developer.android.com/google/gcm/index.html (17.05.2015)

[29] JSON: The Fat-Free Alternative to XML [WWW] http://www.json.org/xml.html (17.05.2015)

[30] Musser, J. (2011). Open APIs: State of the Market [WWW] http://www.slideshare.net/jmusser/open-apis-state-of-the-market-2011 (17.05.2015)

[31] Using OAuth 2.0 to Access Google APIs [WWW] https://developers.google.com/identity/protocols/OAuth2 (17.05.2015)

[32] Ruby programming language homepage [WWW] https://www.ruby-lang.org/en/ (17.05.2015)

[33] Ruby programming language homepage [WWW] https://www.python.org/ (17.05.2015)

[34] Fielding, R. T. (2000) Architectural Styles and the Design of Network-based Software Architectures – *Doctoral dissertation*, *University of California*, 76 [WWW] http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf

[35] Ruby vs. Java benchmark [WWW] http://benchmarksgame.alioth.debian.org/u64q/compare.php?lang=yarv&lang2=java (17.05.2015)

[36] Python 3 vs. Java benchmark [WWW] http://benchmarksgame.alioth.debian.org/u64q/python.html (17.05.2015)

[37] Introduction to the C# Language and the .NET Framework [WWW] https://msdn.microsoft.com/en-us/library/z1zx9t92.aspx (17.05.2015)

[38] Spring framework homepage [WWW] http://projects.spring.io/spring-framework/ (17.05.2015)

[39] Spring boot project homepage [WWW] http://projects.spring.io/spring-boot/ (17.05.2015)

[40] Spring security project homepage [WWW] http://projects.spring.io/spring-security/ (17.05.2015)

[41] Hibernate ORM 4.3 manual [WWW] http://docs.jboss.org/hibernate/orm/4.3/manual/en-US/html/ (17.05.2015)

[42] Jackson JSON processor documentation [WWW] https://github.com/FasterXML/jackson-docs (18.05.2015)

[43] Dagger project homepage [WWW] http://square.github.io/dagger/ (18.05.2015)

[44] Retrofit project homepage [WWW] http://square.github.io/retrofit/ (18.05.2015)

[45] OpenCV library documentation [WWW] http://docs.opencv.org/ (18.05.2015)

[46] Tesseract OCR project homepage [WWW] https://code.google.com/p/tesseract-ocr/ (18.05.2015)

[47] Smith, R. An Overview of the Tesseract OCR Engine [WWW] http://static.googleusercontent.com/media/research.google.com/en//pubs/archive/33418.pdf (18.05.2015)

[48] Smith, R. (2007) Tesseract OCR Engine. What it is, where it came from, where it is going. [WWW] https://tesseract-ocr.googlecode.com/files/TesseractOSCON.pdf (18.05.2015)

[49] Tess4j project [WWW] http://tess4j.sourceforge.net/ (18.05.2015)

[50] Google Play Services [WWW] https://developer.android.com/google/play-services/index.html (18.05.2015)

[51] OAuth 2.0 protocol [WWW] http://oauth.net/2/ (18.05.2015)

[52] RSA algorithm [WWW] http://www.di-mgt.com.au/rsa_alg.html (18.05.2015)

[53] PKCS #12 storage type spec [WWW] https://tools.ietf.org/html/rfc7292 (18.05.2015)

[54] Java concurrency in practice (2006) / Goetz, B., Peierls, T., Bloch, J., Bowbeer, J., Holmes, D., Lea, D., 1st edition, Boston, Addison-Wesley Professional, 105-106

[55] HQL specification [WWW] https://docs.jboss.org/hibernate/orm/3.3/reference/en/html/queryhql.html (18.05.2015)

[56] "Maksukohustuslaste registri" asutamine ja registri pidamise põhimäärus. (2002). – *Riigi Teataja* I, 2002, 68, 408

[57] Estonian e-business register agreement [WWW] https://ariregister.rik.ee/leping.py (18.05.2015)

[58] Estonian e-business register web service specification [WWW] http://www.rik.ee/sites/www.rik.ee/files/elfinder/article_files/XML_p%C3%A4ringute_l%C3%BChikirjeldus_2015_03.pdf (18.05.2015)

[59] FREJ project [WWW] http://frej.sourceforge.net/ (18.05.2015)

[60] Google Developers Console [WWW] https://console.developers.google.com/project (18.05.2015)

[61] Android SQLite database documentation [WWW] http://developer.android.com/reference/android/database/sqlite/SQLiteDatabase.html (18.05.2015)

[62] ActiveMQ messaging server [WWW] http://activemq.apache.org/ (18.05.2015)

[63] SHA1 specification [WWW] http://www.w3.org/PICS/DSig/SHA1_1_0.html (19.05.2015)

Appendix

The source code of the application is available at:

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

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