



**CZECH TECHNICAL
UNIVERSITY
IN PRAGUE**

F3

**Faculty of Electrical Engineering
Department of Computer Science**

Bachelor's Thesis

Mobile application for self-evaluation of skin lesions

Tereza Lemáková

May 2024

Supervisor: Ing. Ivo Malý, Ph.D.

I. Personal and study details

Student's name: **Lemáková Tereza**

Personal ID number: **507222**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Computer Science**

Study program: **Open Informatics**

Specialisation: **Software**

II. Bachelor's thesis details

Bachelor's thesis title in English:

Mobile application for self-evaluation of skin lesions

Bachelor's thesis title in Czech:

Mobilní aplikace pro samovyhodnocení kožních lézí

Guidelines:

Analyze the bachelor thesis [2] that implements detection, classification, and matching of skin lesions. Further analyze the requirements for self-examination of skin lesions using a camera on a mobile phone. Focus on the frequency of examination, the method of photography, and the characteristics of the resulting photographs, as well as the question of security and privacy protection on sensitive photographs. Based on the analysis, create a list of requirements for a mobile client application and also server application using results of thesis [2]. Also, describe user scenarios for using the applications.

Based on the analysis, design the architecture of a mobile client application that will collect user data (photographs) and process them using a server application. Furthermore, design the user interface of the application, at least at the level of a low-level prototype.

Implement mobile client application and also server application based on their designs. For implementation use suitable technologies and frameworks. Implement mobile client application for Android devices.

Test final solution on realistic data/photos.

Bibliography / sources:

[1] Android Developer Portal, <https://developer.android.com/>

[2] Šúr S., Detekce, klasifikace a hledání korespondencí kožních lézí, VUT FEL, bakalářská práce, 2023, <https://dspace.cvut.cz/handle/10467/109023>

[3] Fosu et al.: Mobile melanoma detection application for Android smart phones, NEBEC 2015

[4] Esteva et al.: Dermatologist-level classification of skin cancer with deep neural networks, Nature, 2017

Name and workplace of bachelor's thesis supervisor:

Ing. Ivo Malý, Ph.D. Department of Computer Graphics and Interaction FEE

Name and workplace of second bachelor's thesis supervisor or consultant:

Date of bachelor's thesis assignment: **02.02.2024** Deadline for bachelor thesis submission: **24.05.2024**

Assignment valid until: **21.09.2025**

Ing. Ivo Malý, Ph.D.
Supervisor's signature

Head of department's signature

prof. Mgr. Petr Páta, Ph.D.
Dean's signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce her thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

Date of assignment receipt

Student's signature

Acknowledgement / Declaration

I want to thank my supervisor Ing. Ivo Maly Ph.D. for his support and patience during the preparation of this thesis.

I would also like to thank my family and friends for their support during my studies, and all my testers and models for making this project possible.

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

In Prague, 24 May 2024

.....

Abstrakt / Abstract

Tato práce se zaměřuje na vytvoření dostupného nástroje pro sekundární prevenci rakoviny kůže. Aplikace je rozšířením práce [1] a používá vytvořený algoritmus založený na konvoluční neuronové síti pro detekci a vyhodnocení obrázků kůže.

Cílem je poskytnout podrobnou analýzu požadavků a přínosů samovyšetření kůže a technik fotografování, které přispívají k dosažení co nejpřesnějších výsledků konečné aplikace.

Na základě analýzy jsem navrhla a vyvinula nativní aplikaci pro Android s jednoduchým a srozumitelným uživatelským rozhraním, implementovaným pomocí frameworku Jetpack Compose, a serverovou Spring Boot aplikaci, která zprostředkovává ukládání dat a vyhodnocování pomocí algoritmu.

Dále bylo provedeno testování uživatelské přívětivosti aplikace, které přineslo převážně pozitivní zpětnou vazbu. Na základě odpovědí účastníků jsem navrhla úpravy pro zlepšení aplikace a implementovala některá z nich.

Klíčová slova: prevence rakoviny kůže, mobilní aplikace, Android, CNN, architektura klient-server, Spring Boot, Jetpack Compose, e-health

This thesis focuses on creating an accessible tool for secondary prevention of skin cancer. The application is an extension of the thesis [1] and uses the created convolution neuron network (CNN) based algorithm to detect and evaluate images of skin.

It aims to provide a detailed analysis regarding requirements and benefits of self-assessment of skin, and techniques of photography that can be used to get the best result from the final application.

Based on the analysis, I designed and developed a native Android client application with a simple and comprehensible user interface implemented using Jetpack Compose framework, and a Spring Boot server application that provides data storage and runs the CNN algorithm.

Furthermore, usability testing was conducted and yielded mostly positive feedback from the users. Based on the testers' responses I suggested improvements to the application and implemented some of them.


Keywords: skin cancer prevention, mobile application, Android, CNN, client-server architecture, Spring Boot, Jetpack Compose, e-health

Contents /

1 Introduction	1	4.3 Server.....	23
1.1 Objective	1	4.3.1 Languages and Technologies.....	23
1.2 Structure	1	4.3.2 Project Structure	24
2 Analysis	3	4.3.3 Authentication	24
2.1 Prevention and Screening... 3		4.3.4 Authorisation	25
2.1.1 Identifying High-Risk Groups.....	3	4.3.5 REST API Definition	27
2.1.2 Visual Self-Examination	3	4.3.6 Data Storage	28
2.1.3 Technology-Assisted Self-Examination	4	4.4 Client.....	29
2.2 Existing Algorithm	5	4.4.1 Language and Technologies.....	29
2.2.1 Inputs	5	4.4.2 Project Structure	30
2.2.2 Outputs	6	4.4.3 Security	30
2.2.3 Diagnosis Interpretation.....	7	4.4.4 User Interface Definition	31
2.2.4 Drawbacks	8	5 Testing	35
2.3 Photography	9	5.1 Usability Testing.....	35
2.3.1 User Study	10	5.1.1 Questionnaire	35
2.4 Functional Requirements ..	11	5.1.2 Observations	37
2.5 Use-Cases	11	5.1.3 Summary	38
2.5.1 Login/Registration... 11		5.2 Development Testing	38
2.5.2 Create Single Record.....	11	6 Conclusion	41
2.5.3 Create New Record from Existing (Update a Record).....	12	6.1 Results	41
2.5.4 Create a Complete Mapping.....	13	6.2 Possible extensions.....	41
2.5.5 Discontinue Record (Mark Record as Outdated).....	14	References	43
3 Design	15	A List of Abbreviations	47
3.1 Datamodel	15	B List of Electronic Appendices	48
3.2 Architecture.....	16	B.1 Source Code.....	48
3.3 Server.....	17	B.2 User Manual	48
3.3.1 Database	17	C Placement Photography List	49
3.3.2 Interface.....	17		
3.4 Mobile Client.....	18		
3.4.1 User Interface	18		
4 Implementation	23		
4.1 Security.....	23		
4.2 Project Management.....	23		

Listings / Figures

4.1. Okta dependency to enable Auth0 on the server... 25	2.1. Example output of the CLI application 6
4.2. Customised Filter Chain definition 25	2.2. Example output.csv file 6
4.3. Definition of the list of protected URLs 25	2.3. Original result cut-out 7
4.4. Method for photographs access authorisation 26	2.4. Modified result cut-out 7
4.5. Method for Mapping access authorisation 26	2.5. False positive identification example..... 7
4.6. Method for PhotoRecord access authorisation 26	2.6. Example of failed detection of bigger lesions 8
4.7. Example of the usage on preAuthorize annotation... 26	2.7. Back and front camera image quality comparison... 9
4.8. Auth0 dependencies for working with the universal login page..... 31	2.8. Illustrative image from the photography study..... 10
4.9. Definition of the client AuthInterceptor 31	3.1. Diagram of the general datamodel 16
5.1. Example of Mockito usage in testing AuthorizationService..... 38	3.2. General client-server architecture diagram 16
5.2. Example of tests for authorisation accuracy 39	3.3. Lo-Fi design of My Body screen 19
	3.4. Lo-Fi design of History screen 19
	3.5. Lo-Fi design of Settings screen 19
	3.6. Lo-Fi design of the Show Record screen 20
	3.7. Lo-Fi design of the New Record screen 20
	3.8. Lo-Fi design of the New Mapping screen..... 21
	3.9. Lo-Fi design on the Log In screen 21
	4.1. Example of using DBRef... 28
	4.2. Example of combining different referencing methods 29
	4.3. Example of storing incomplete entities 29
	4.4. Screenshots of My body screens 32
	4.5. Screenshot of the record overview 32
	4.6. Screenshot of the record overview with detected lesions..... 32
	4.7. Screenshot of the record info dialogue..... 33



4.8. Screenshot of the related records review	33
4.9. Screenshot of the starting screen	33
4.10. Screenshot of the login page	33

Chapter 1

Introduction

According to the International Agency for Research on Cancer, which is a part of the World Health Organization, around 1.5 million new cases of skin cancer were diagnosed worldwide in 2022.¹ This makes skin cancers² the most common group of cancers in the world. Unfortunately, professionals expect the number of people suffering from this type of condition to only increase in the future [2].

Early diagnosis of these conditions can vastly decrease the severity and the mortality. Technological advancements in e-health, particularly through mobile applications, offer a promising solution to this challenge. It can provide accessible, efficient, and reliable tools for regular self-examination, potentially leading to earlier detection and better outcomes.

1.1 Objective

The primary goal of this thesis is to conduct a detailed analysis and design of a simple yet efficient tool for mapping and screening for potentially harmful skin conditions. The result should be a comprehensive native Android application.

The format of a mobile application was selected, because according to the *Global System for Mobile Communications Association* [3], as of 2023 a majority of people globally own smartphones. The reason for selecting Android as the primary platform for this project is it is generally more commonly used [4]. Additionally, during the research was discovered that there are several applications with similar premises available for iOS devices.

The uniqueness of this application lies in the fact, that it not only provides a tool to store and compare skin lesions for a longer time, but it also uses a CNN algorithm to evaluate presented photographs. This algorithm can classify not only melanomas but also other types of potentially harmful lesions [1]. It also provides a detection algorithm, that enables bigger sections of skin to be analysed at once. (This topic is elaborated in sections 2.1.3 and 2.2.2.)

1.2 Structure

This work is composed of six main chapters the first one being the introduction. The second chapter is dedicated to the analysis of the topics related to the application. It regards both the medical and technical side of the project as it provides insight into skin cancer prevention and screening, analysis of the used algorithm [1], and techniques of photography. The third chapter discusses the design of the application. It specifies the selected architecture of the application and describes the interface of the server and the user interface

¹ <https://www.iarc.who.int/cancer-type/skin-cancer/#summary>

² in context of including both melanoma and nonmelanoma skin cancers

(also called just UI) of the Android client. The fourth chapter summarizes the technologies and approaches chosen for the development of both parts of the application. Chapter five mostly describes the user testing and the sixth chapter provides a summary, a comparison with the original assignment, as well as some suggestions of possible future extensions for the implemented application.

Chapter 2

Analysis

This chapter provides a complex analysis regarding general skin cancer prevention, provided CNN algorithm from [1], and techniques of photography.

2.1 Prevention and Screening

WHO defines primary prevention as a set of actions aimed at avoiding the manifestation of a disease, whereas secondary prevention deals with early detection and improves the chances for positive health outcomes [5].

In association with skin cancer, an example of primary prevention can be limiting exposure to UV rays by covering up or using sunscreen. Regular check-ups with a dermatologist can be considered a secondary preventative measure.

Annual professional skin checks supported by self-checks every 2 to 3 months are recommended to most of the general population. However, for certain people, this frequency of screenings might be excessive. On the other hand, individuals with elevated risks of skin cancer might benefit from more frequent visits.

2.1.1 Identifying High-Risk Groups

Many factors can elevate the risk of skin cancer. Some of them are non-modifiable such as age over 40, skin phototype I – II¹, or family and personal history of skin cancer. Other factors are environmental, these might include for example immunosuppression (most frequently associated with an organ transplant), usage of solariums [6], jobs necessitating a higher sun exposure (e.g. construction workers, farm workers), or jobs with higher exposure to UV radiation (e.g. flight attendants, radiology workers) [7].

People who are part of these high-risk groups might benefit from more frequent self-checks, such as once every month.

2.1.2 Visual Self-Examination

Self-examination is a useful tool for secondary prevention that is available to anyone. The techniques generally used in self-assessment include the inspection of the whole body with the use of mirrors.² If a suspiciously looking lesion is identified, the individual is advised to seek a professional opinion.

However, the distinction of a suspicious lesion might not be so straightforward. One of the most well-known methods is called the ABCD rule that has been originally introduced in 1985 [8] and can be used to distinguish between benign and malignant skin lesions. The rule was originally based on the asymmetry, border irregularity, colour, and differential structure of the lesion [9].

¹ <https://dermnetnz.org/topics/skin-phototype>

² <https://www.cancer.org/cancer/risk-prevention/sun-and-uv/skin-exams.html>

Most modern sources use an extended version of this rule, called the ABCDE, where D is changed to diameter and E stands for evolution. Optionally, even the letter F can be added, which denotes *Funny looking*, meaning that the most suspicious lesions are those with atypical appearance [8].

The main drawback of self-examination is that in healthy people it can easily result in misdiagnosis and overtreatment. It is also insufficient as a preventative measure, if not combined with professional diagnosis. For this, the effectiveness of this practice is controversial [10].

2.1.3 Technology-Assisted Self-Examination

Using a smartphone application is an easy and accessible way to improve the quality and credibility of self-examination. There are many approaches to how to utilise smartphones in skin mapping and cancer prevention.

Some applications connect users with medical professionals who help them diagnose the conditions from photographs. An example of this service is **iDoc24**¹ or **First Derm**². These applications can be especially beneficial for people with mobility issues and people with limited access to specialised facilities (for example due to distance). The main downside of this type of service is that it typically does not provide the doctor with sufficient background on the patients' health. This may result in a lengthy diagnosis or sometimes even misdiagnosis [10]. The usage of these applications is also usually quite costly, as the users are effectively seeking the help of a private medical professional. This might make this service inaccessible for some people.

Other applications focus on the importance of regular skin checks. These applications typically provide a place to store and compare images of skin for a longer time. They don't usually provide any kind of analysis or diagnosis of the lesions. An example of such an application is **MoleMapper**³. This type of application is beneficial because it stores user data for future review. However, it does not automate any processes, and if the user has multiple suspiciously evolving areas on the body, the process of reviewing the past records can be lengthy and tiring, especially for elderly people.

The last notable approach is the usage of *Artificial Intelligence* (referred to as AI) to classify the lesions. An example of this approach might be **curesskin**⁴ application which focuses on diagnosing a broader spectrum of skin and hair conditions with an AI tool and suggests medical treatments as well as dietary changes and supplements. A different type of AI is used for example by the application **AI Dermatologist**⁵ which provides a tool to assess close-up images of a single lesion. The main problem with this technology is that AI is not yet accurate enough to make a reliable diagnosis. This can cause misdiagnosis which can result in unfounded stress in the users.

Some applications combine the first two mentioned approaches as they provide tools for continuous mapping with the possibility of submitting an image to be seen by a medical professional. This is, for example, the **Miiskin**⁶ applica-

¹ <https://idoc24.com/>

² <https://www.firstderm.com/>

³ <https://molemapper.org/>

⁴ <https://curesskin.com/>

⁵ <https://ai-derm.com/>

⁶ <https://miiskin.com/app/>

tion which contains an advanced imaging system to improve the photo quality and make the process as easy as possible.

In this thesis, I am aiming to develop an application that will combine the latter two approaches by evaluating all created images with AI and providing the user with the ability to create and continuously update records of the skin (photographs), and see their progression in time.

2.2 Existing Algorithm

One of the main features of the application is going to be the detection and classification of skin lesions using the CNN algorithm. To be able to integrate the preexisting algorithm into the application it is necessary to thoroughly analyse the inputs and outputs of the Python pipeline.

The aim is to be able to modify the code of the original algorithm as little as possible to prevent any possible inconsistencies while receiving an output that is as compatible with the new application as possible. To use the provided code to the fullest extent, I will be using the command line interfaced (usually referred to as CLI) application that simplifies access to the algorithm [1].

We are also presented with the question of how should the final output of the algorithm be presented to the end user. This quite sensitive topic is expanded on in section 2.2.3.

2.2.1 Inputs

The algorithm itself takes an input of a NumPy array in RGB colour format. But CLI application needs a path of an image in the .jpg or .jpeg format. The requirements for the provided image are stated in the [1] as follows:

- Minimal resolution of the image is at least *24 mp*.
- The quality of the image is at least *60 points* of the BRISQUE score.

In the original CLI application, this image assessment is a part of the process of lesion detection. To increase the usability and accessibility of the new application it was decided to divide these processes. When the image quality and resolution are evaluated first, the user does not have to wait for the lesion detection and classification to finish, before using the application further.

1. Resolution

The resolution of the provided image can be easily determined in code. Because some smartphone cameras, especially on older devices, might not be able to provide images with the required resolution, the photographs can be programmatically resized. This can be achieved by using functions provided by an external library for altering photographs, such as Java Image-Scaling Library *imgscalr*¹.

2. Quality

In the original implementation, the quality check is provided by the Python PIQ library. It uses the BRISQUE algorithm, that provides a *no-reference*² image quality assessment [11]. During the research, I could not find any suitable libraries for Java or Kotlin, that would provide the desired quality

¹ <https://github.com/rkalla/imgscalr>

² also called *object blind*

evaluation. Because of that, I suggest the assessment be provided by a Python script, run on the server side of the application.

2.2.2 Outputs

The CLI application provides multiple outputs. For compatibility with the new application, it needs to be determined which outputs will be used and how will those need to be modified.

As of now, all the newly created files are being inserted into a `/results` folder. When calling the detection and classification script from the Java code, the `result` folder will be created in the base project folder.

When using the provided application, the main output one would use is what is displayed in the command line. As can be seen in Figure 2.1, the formatting of the text output is clear and well-arranged to be legible by the human eye. However, it contains multiple spaces between each value and other graphical features. Therefore, it would be rather challenging to parse between the programs. It also duplicates the data, presented in other possible sources.

For these reasons, the command line output will not be used in this case.

```

$ python main.py -i "C:\Users\Tereza\Desktop\FEL\BP\bakalarska_prace\server\1.jpg"
100% |
image_name  detection_scores  NV      MEL      BCC      BKL      AK      SCC      DF      VASC      probs  unknown  preds
0  box_0            0.372873  0.355451  0.026537  0.069731  0.027902  0.022845  0.002271  0.632970  0.056240  0.632970  False  DF
1  box_1            0.318663  0.743986  0.097259  0.028379  0.067138  0.015687  0.001044  0.040546  0.005961  0.743986  False  NV
2  box_2            0.244576  0.074559  0.007177  0.015125  0.008546  0.001996  0.002031  0.887041  0.003532  0.887041  False  DF

```

Figure 2.1. Example output of the CLI application

For easier referencing in this text, the other outputs will be divided into three parts CSV file, image cut-outs, and modified original image.

1. CSV File

One of the outputs of the CLI application is a file named `output.csv`. This file contains a header line and then a line for each lesion that has been detected. This provides a strong and easily parsable source of the data that needs to be extracted from the pipeline. An example of the contents of the `output.csv` file can be seen in Figure 2.2.

Specifically, information that is relevant for the user is the image of the lesion (the first value on each line), whether the lesion was able to be identified (the 12th value), and in case the identification was successful, how it was classified (the 13th value).

```

image_name,detection_scores,NV,MEL,BCC,BKL,AK,SCC,DF,VASC,probs,unknown,preds
box_0,0.37307268381118774,0.15545143,0.026587114,0.0697307,0.027901944,0.022844646,0.008270507,0.6329699,0.05624368,0.6329699,False,DF
box_1,0.31866297125816345,0.74398607,0.097259104,0.028378524,0.067138314,0.015686614,0.0010441188,0.040546365,0.0059609115,0.74398607,False,NV
box_2,0.24457575380802155,0.07454964,0.0071772654,0.015125461,0.008546383,0.0019963647,0.0020309906,0.88704145,0.003532481,0.88704145,False,DF

```

Figure 2.2. Example output.csv file

2. Image Cut-outs

Another output of the algorithm is a series of cut-outs of the original images, containing the detected lesions. In the original form, these cut-outs are completed with the final classification and the percentage regarding the probability of the classification.

For reasons that are expanded on in section 2.2.3, the end users cannot be presented with a precise diagnosis, as is presented in the original image.

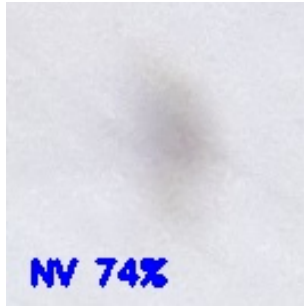


Figure 2.3. Example of an original cut-out.



Figure 2.4. Example of a newly modified result cut-out.

Removing the label from the cut-out will also help with the legibility of the image, as can be seen in Figures 2.3 and 2.4.

3. Modified Original Image

The last output of the Python script is an image with drawn-on boxes, of the detected lesions. This form of output is not going to be very legible to the user due to the size of the detected lesions. However, it is going to be important in identifying incorrectly detected lesions¹ (as is demonstrated in Figure 2.5). These are formations that were misinterpreted by the detection algorithm as a skin lesion, while not being a part of the users' skin.



Figure 2.5. Example of how the modified image can be used to identify incorrectly detected lesions.

2.2.3 Diagnosis Interpretation

The AI algorithm can yield one of the following classifications: *AK*, *BCC*, *NV*, *BKL*, *SCC*, *DF*, *MEL*, *VASC*, or Unknown. All the presented abbreviations are associated with a certain type of lesion. However, even if the users are presented with full-name classification, most of the general population will not know the ramifications of the diagnosis. The results of the provided algorithm are also not medically proven to be accurate.

¹ also called *False Positives* [1]

The goal is to make the application output as comprehensible as possible, whilst refraining from misdiagnosing users with some severe medical conditions such as skin cancer. For these reasons, it was decided to divide these classifications into generalised groups corresponding to harmless, suspicious, and unrecognised. Lesions that were marked as suspicious are mostly those classified as cancerous or potentially precancerous.

The presented classifications were decided to be mapped as follows:

- Actinic keratosis (AK): Suspicious [12]
- Basal cell carcinom (BCC): Suspicious [13]
- Melanocytic nevus (NV): Harmless [14]
- Benign keratosis-like lesions (BKL): Harmless [15]
- Squamous cell carcinoma (SCC): Suspicious [16]
- Dermatofibroma (DF): Harmless [17]
- Malignant melanoma (MEL): Suspicious [14]
- Vascular lesion (VASC): Suspicious [18]

■ 2.2.4 Drawbacks

During the testing of the algorithm behaviour, it was discovered that there were some issues with the outputs. These matters mainly concern lesion detection and they might hinder the usability of the final product.

The first problem is the **inability to detect bigger lesions**. As can be seen in Figure 2.6 the algorithm fails to detect the most obvious lesions on the image. Even if it is specified in the User Manual (appendix B.2) that wider areas should be photographed instead of close-ups on single lesions, users are still the most interested in analysing the prominent lesions.



Figure 2.6. Example of failed detection, comparison of detected lesions (blue square) and most prominent lesions (red circles)

The other downside of the algorithm usage is so-called **false positives**. This topic was already touched upon in the section 2.2.2 where it was discussed how to help users identify them. In the User Manual (appendix B.2), users are advised on which kinds of backgrounds are not ideal for the photographs. This way the number of incorrectly detected lesions can be limited, but they can not be eliminated.

2.3 Photography

Taking photos of skin is a crucial part of mapping skin conditions. For some people, this might involve taking pictures of body parts that are not easily accessible. If these images are not taken properly, this might result in undetected or misdiagnosed lesions.

This section will explain what are some good and bad practices in photography. To relay this information to the user, I use a User Manual (provided in appendix B.2)

Modern smartphones are mostly made with front and back cameras. If a smartphone owner is asked to take a photo of their face, they would intuitively use the front camera as it is more comfortable than the back camera. The reason why this kind of behaviour is not supported by the application is that front cameras typically provide lower resolution [19]. This can be seen in Figure 2.7. As mentioned in section 2.2.1, it is important that the resolution and quality of the image are maximised.

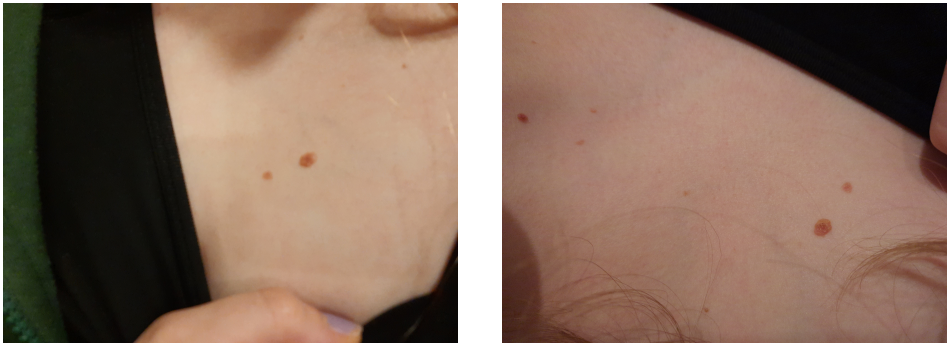


Figure 2.7. Comparison of quality and resolution between a photo of the same placement taken by the front camera (left) and back camera (right).

Some basic tips regarding taking photos of the skin are:¹

- **Clean your lens.** – Wipe your phone’s camera lens/es off with a wipe of a cotton piece of cloth before starting your mapping.
- **Use back camera.** – With most smartphones, the quality of photos, taken by the back camera is a lot better than the selfie/front camera.
- **Remove your phone case.** – If you have a closable phone case, it can get in the way of you comfortably holding your phone. Removing it might help you get a better grip on your phone.
- **Ask for help.** – Some places may be hard to reach, especially for people with a limited range of motion. If possible, ask your friend, relative, roommate, or any other person to help you with the photography.
- **Photograph wider areas.** – Taking a close-up photo of a single lesion seems like a good idea, but it might not be the best. The AI used for this application works better when a bigger area of skin is presented. It is also easier to spot any changes in the size and shape of a lesion when you can compare it to its surroundings.

¹ These tips have been mostly compiled during user testing and interviews. The formulation is straight taken from the manual B.2.

- **Adjust the light.** – Remember to take your photos in a well-lit area. If it is dark outside, turn the light on. But be aware that deeper shadows can make the AI analysis less accurate.
- **Be aware of your surroundings.** – Darker spots in your background can confuse the AI. Reconsider taking photos with dotted or wooden surfaces as those are likely to be interpreted as lesions.
- **Take your time.** – Taking a lot of photos in a quick succession might be tiring and the quality can increase during the mapping. Remember that you can always take the pictures in advance, even during various hours or days, and then upload them into the application.

■ 2.3.1 User Study

A test was conducted on 3 able-bodied individuals ages from 17 to 22. They were provided with a list of parts of their bodies (as can be seen in C), an Android smartphone, a wall mirror, a selfie stick¹, and a chair. The participants were asked to use the back-facing camera of the device to take photos of all the body parts on the list. The progress of their work was observed and documented and the quality of taken photographs was monitored.²

Based on the outcome, it was determined that people with high mobility are capable of photographing most of their bodies on their own. The only places that were determined to be inaccessible are the lower back, buttocks, back of the neck and some places on the head³.

In Figure 2.8 a participant is using a combination of the mirror and the selfie stick to take a photo of the back of their tights.



Figure 2.8. An image of a participant trying to take a photo in a difficult location taken during the photography study.

¹ compatible with the provided device

² All the participants agreed to take photographs during the study, and for these photographs to be used in this thesis.

³ depending on how much and how long hair one has

2.4 Functional Requirements

- **FR01.** The user should be able to create a new user account.
- **FR02.** The user should be able to log into an existing account.
- **FR03.** The user should be able to create a new record containing an image and a description.
- **FR04.** The user should be able to create a new record containing an image but no description.
- **FR05.** The user should be able to create a record using an existing image.
- **FR06.** The user should be able to create a record by taking a new photograph.
- **FR07.** The user should be able to see the image of the evaluated record.
- **FR08.** The user should be able to see the result of the evaluation of a submitted image.
- **FR09.** The user should be able to see detected lesions.
- **FR10.** The user should be able to compare images of two or more related records.
- **FR11.** The user should be able to update the image for a selected record.
- **FR12.** The user should be able to comprehensively create a new mapping (meaning, create an updated version of all up-to-date records in the current mapping).
- **FR13.** The user should not be able to access data (photographs, records, mappings etc.) that were not created by them and are not linked to their account.

2.5 Use-Cases

This section provides specific use cases and scenarios to describe the intended usage of the application.

2.5.1 Login/Registration

Precondition: The application has been successfully downloaded from a supported provider and installed.

1. The user opens the application.
2. The system displays declarations and asks the user if they wish to accept and continue.
3. IF the user confirms the usage, the system redirects to the *centralised login page* provided by a third party. IF the user does not wish to continue they may leave the application.
4. IF the login/registration process is successful, the system is notified by the third party and the user is redirected to the home page of the application.

2.5.2 Create Single Record

Possible scenarios:

- The user has been suggested/decided to start using the application and they want to start mapping their skin.
- The user of the application has noticed a possibly suspicious lesion on a part of their body they are not yet mapping via the application.

- A certain part of the user's body has become more risky (for example they sunburned some section of their body), so they want to monitor it more closely and more frequently.

Precondition: The user has previously created at least one record as specified in 2.5.2.

1. The user opens the application and selects the record they wish to create an update. The user selects the option to create a new record from an existing one.
2. The system displays the form to create a new record with preselected placement and prefilled description (if any is present in the original record).
3. Optionally, the user can update the record description.
4. The user follows to add an image as specified in 2.5.2 points 4 – 9.
5. After the evaluation, the system removes the original record from the mapping and substitutes it with the updated one.
6. The user can (but does not have to) navigate to see the results of the photo evaluation. They can also navigate to see the comparison between the previous and current photos.

■ 2.5.4 Create a Complete Mapping

Possible scenarios:

- The user was notified that the time (usually 2 to 3 months) has passed and they should create a new complete mapping.
- The user was/is about to be subjected to some behaviour that increases the risk of skin cancer and wants to create a new mapping of their body so they could monitor the evolution of their skin more closely from that point.

Precondition: The user has previously created at least one record as specified in 2.5.2.

1. The user opens the application and chooses the option to create a new mapping.
2. The system displays a screen with instructions on how to create a mapping successfully and the first record to replicate
3. User reads the instructions and either decides to replicate the record or to skip it.
4. IF the user chooses to skip the record, the process continues at point 6. IF the user chooses to replicate the record, they are presented with a prefilled form for record creation, which they fill in as specified in 2.5.2 points 4 – 8.
5. IF the quality of the image is insufficient, the system displays a dialogue and the user is prompted to retake/choose a new photo. ELSE the process continues at point 6.
6. The system creates the new record locally. IF there are any more records to be replicated, the system displays a preview of the following record. And the process resumes at point 3. ELSE the process continues at point 7.
7. The system sends the list of new records to the server to be stored and evaluated.
8. After the evaluation of all the new records a new mapping is created and displayed on the home page.

9. The user can (but does not have to) see the newly created mapping and review the records.

■ 2.5.5 Discontinue Record (Mark Record as Outdated)

Possible scenarios:

- The selected area has not had any development in a very long time so the user wants to stop mapping it.
- Because of some external reason (e.g. lesion removal, skin injury) mapping the selected area is no longer relevant or necessary for the user.

Precondition: The user has previously created at least one record as specified in 2.5.2.

1. The user opens the application and navigates to a selected record. The user chooses to discontinue the record.
2. The system marks the record as discontinued and visually removes it from the current mapping.
3. The user can access a discontinued record via the history tab. IF the user wants to update the record even after it has been discontinued, they can choose to update the archived record.

Chapter 3

Design

This chapter provides a detailed design of the application. It contains a definition of the data model, describes the intended architecture, shared interface, and *Low Fidelity* (referred to as Lo-Fi) design of the *User Interface* (also called UI).

3.1 Datamodel

In this section, I define basic entities that are in some variations going to be used in implementation. The UML diagram is shown in Figure 3.1.

■ Mapping

- Defines a specialised collection for objects of the `PhotoRecord` datatype. As a whole, it defines one user's skin mapping in a certain period of time.
- **Attributes:** unique identifier, note (optional)
- This class provides getters ¹, a method for adding one or multiple records, retrieving dates when the oldest and the newest record in the collection was created, and a method to retrieve the AI result of the complete mapping.

■ PhotoRecord

- Represents one record of an image.
- **Attributes:** UID, compressed image, placement (defined by the enumerative type), String note for placement specification, date of record creation, number of related records, list of `PhotoSpecifications`
- Provides a method to get the analysis result of the record.

■ PhotoSpecification

- Represents an image cut-out (as defined in section 2.2.2) combined with the result of the AI analysis.
- **Attributes:** AID, AI result, image cut-out containing a single lesion

■ Enumerated types:

- **Placement** – specifies the general placement of the image
Values: chest, stomach, upper back, lower back, buttocks, right upper arm, right lower arm, right hand, left upper arm, left lower arm, left hand, right thigh – front, right thigh – back, right calf, right shin, right foot, left thigh – front, left thigh – back, left calf, left shin, left foot, neck, face, head
- **AIResult** – specifies the result of the AI analysis of the selected lesion (as define in section 2.2.2)
Values: ok, unrecognised, suspicious

¹ methods for retrieving attributes in a safe and controlled manner

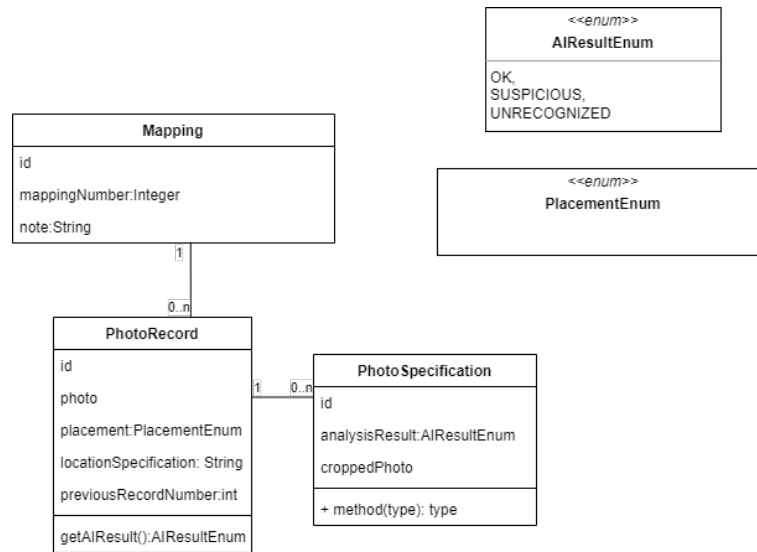


Figure 3.1. Diagram of the general data model

3.2 Architecture

This project aims to create a mobile application that stores and evaluates user images. However, the evaluation process is rather lengthy and storing many high-quality photographs can be memory-intensive. Therefore, this project is going to use the **Client-Server architecture**.

Client-server architecture typically features multiple users' workstations, PCs, or other devices, connected to a central server via an Internet connection or other network. (As is illustrated in Figure 3.2.) The client sends a request for data, and the server accepts and accommodates the request, sending the data back to the user who requested them [20].

In this case, the server mainly going to store the data and provide operations with images. The client, which is to be represented by an Android application, is going to fetch and display selected data and collect data from the users.

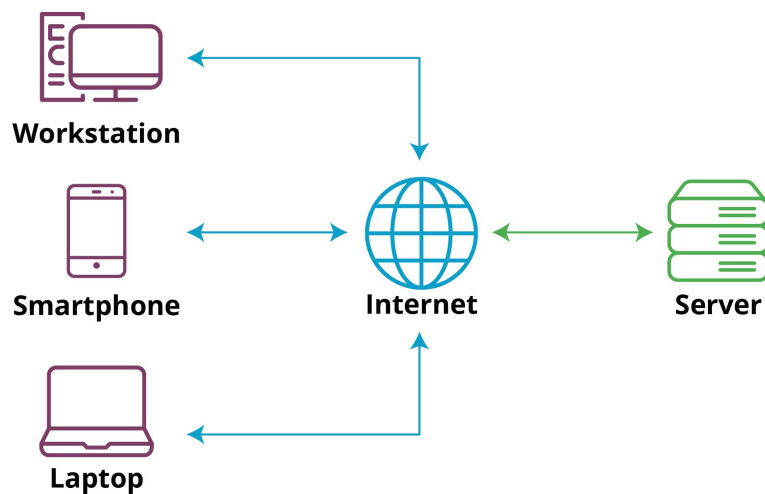


Figure 3.2. An illustrative diagram of client-server type architecture taken from [21]

3.3 Server

The main task of the server application is to store and evaluate user data. In this section, both of these goals will be explained and defined.

3.3.1 Database

When it comes to storing data on a server, there are two main options. You can either use a **SQL** database (such as *PostgreSQL*) or **NoSQL** database (such as *MongoDB*).

Many things need to be taken into consideration when choosing how to store your data. In this case, the main indicator is going to be the type of data that is going to be stored. Looking at the general data model in section 3.1, it is apparent that most of the data used in this application is heavily nested and would be rather challenging to store in a table. Another thing that needs to be taken into account is that images need to be stored on the server.

Seeing that it is rather complicated to use SQL databases to store unstructured data (as explained in [22] and [23]), this project is leaning towards a NoSQL database.

3.3.2 Interface

To communicate with the client application, the server needs to implement an Application Programming Interface usually referred to as API. This interface needs to provide the following features:

■ Image quality assessment

Evaluating image quality (as specified in section 2.2.2). After the image is evaluated it is also saved into the database.

Input: image in the JPEG format

Output: image quality assessment result (true/false), image UID

■ Single record creation

Evaluating a single record with AI algorithm and saving it into the current mapping. This process requires some time.

Input: corresponding image UID, placement specifications

Output: record UID

■ Creation of a related record

Creating a record that is to be marked as related to an existing record.

Input: existing record UID, UID of the image that the record is being created for, placement specification (the default value here is the same as the existing record)

Output: new record UID

■ Multiple record creation

Evaluating a list of records that are all related to existing records. This process should also create a new mapping.

Input: list of image UIDs, placement specifications, and related records' UIDs

Output: UID of the new mapping

■ Reevaluate record

Rerun the AI analysis and update the selected record.

Input: UID of the record to be reevaluated

No output is necessary here.

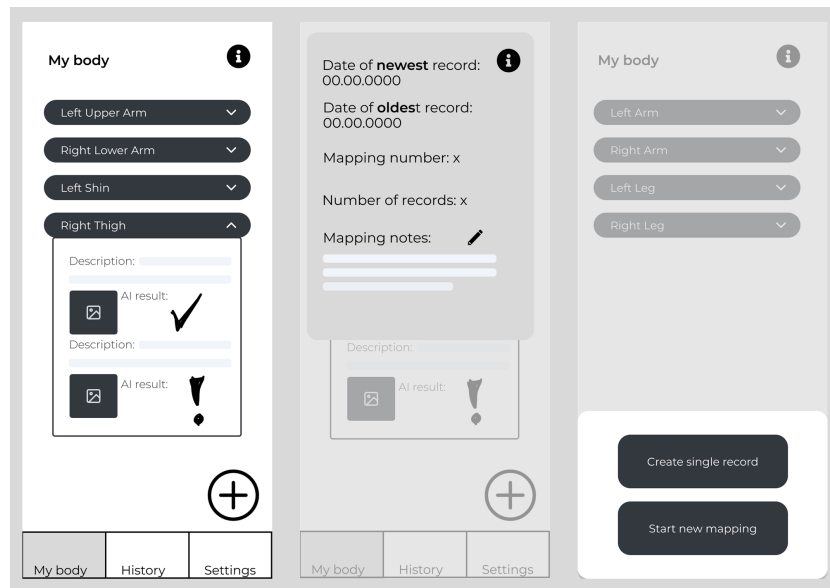


Figure 3.3. Lo-Fi design of My Body screen

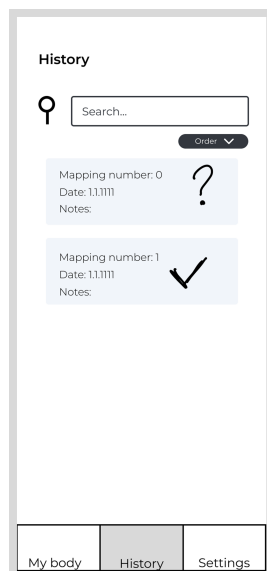


Figure 3.4. Lo-Fi design of History screen

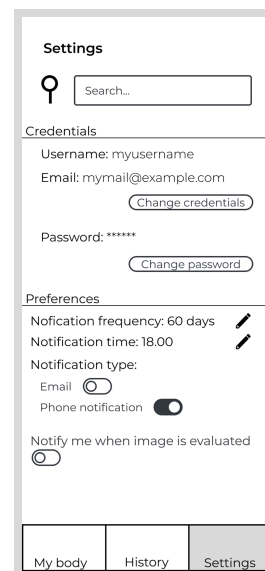


Figure 3.5. Lo-Fi design of Settings screen

In case the user wants to compare multiple images of the same location, they can use the button in the detailed information dialogue to see the list of images from the related records. (As can be seen in the rightmost frame of Figure 3.6.) This feature provides a fast and intuitive option to study lesions over time.

This page also provides buttons for discontinuation (as defined in use-case 2.5.5), rerunning the analysis and creating a related record (updating the image of an existing record).

■ New Record

This page provides a form to create a new single record (as defined in use-case 2.5.2). It is composed of a drop-down menu, containing a list of predefined general areas, and a text box for other notes and descriptions as

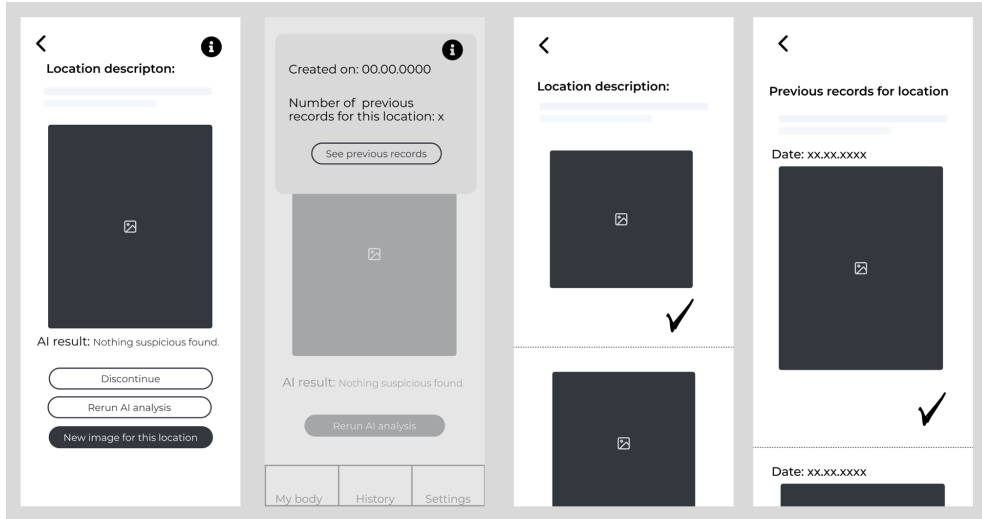


Figure 3.6. Lo-Fi design of the **Show Record** screen and the specification list

is depicted in Figure 3.7. In case the record is created from an existing one, these fields are prefilled based on the original record. There are two options to add an image to the record, the user can either use an image from the gallery or take a new picture (as defined in FR5 and FR6 in section 2.4).

If an image has been added a button for image evaluation is displayed as is shown in the second frame from the left in Figure 3.7. In case the image quality is not determined to be sufficient (based on the rules defined in section 2.2.1) a dialogue notifying the user is to be displayed. The design of the dialogue is presented on the second frame from the left in Figure 3.7.

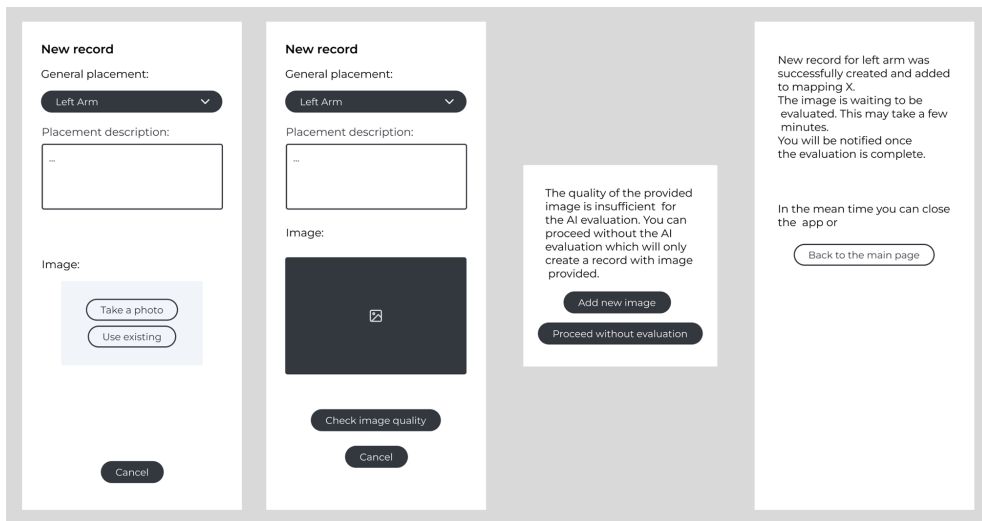


Figure 3.7. Lo-Fi design of the **New Record** screen

■ New Mapping

In case the user wants to start a new mapping they are met with a guide on how to proceed with recreating images. The process follows the use-case defined in 2.5.4. The notes on previews of the records encourage the user to try to replicate the existing photographs as closely as possible as can be seen in both frames of Figure 3.8.

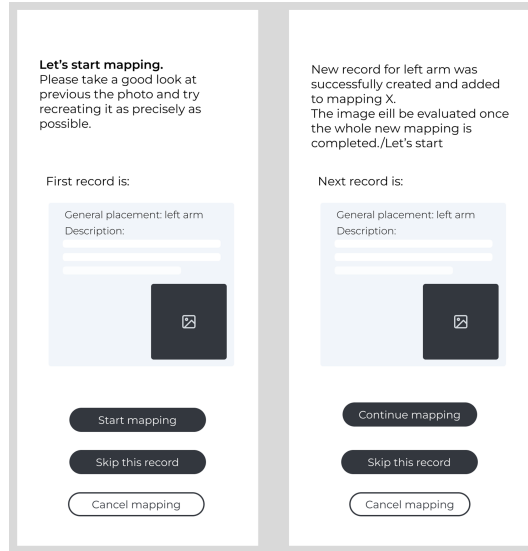


Figure 3.8. Lo-Fi design of the **New Mapping** screen

■ **Log In**

The design of the login page helps the user to intuitively create an account or sign in. The screen with the form to create an account also contains a disclaimer that the application is not a reliable source and it is for preventative purposes only (this can be seen in the middle frame of Figure 3.9).

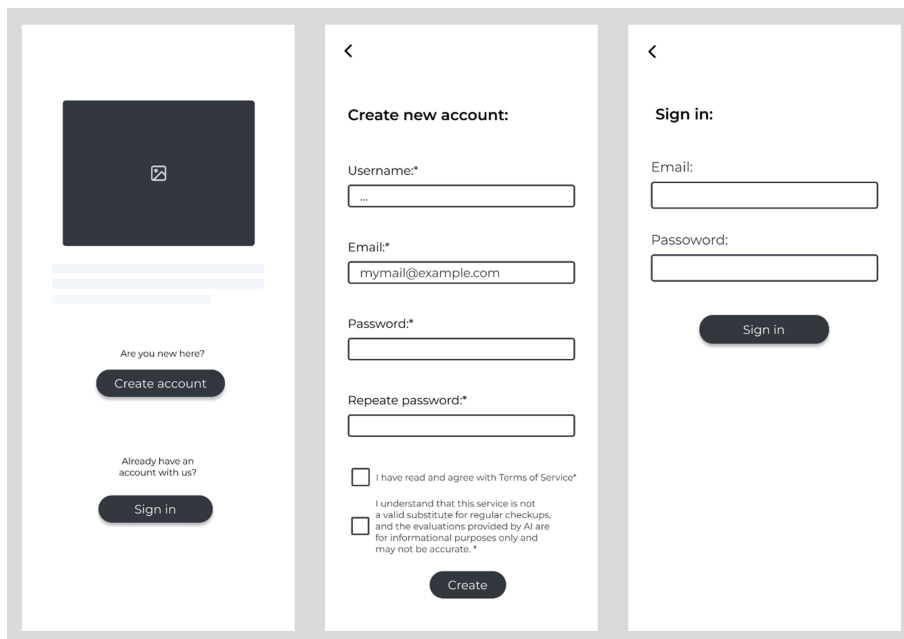


Figure 3.9. Lo-Fi design on the **Log In** screen

Chapter 4

Implementation

This chapter provides insight into how the application was developed. As explained in the section 3.2, this project is following the client-server architecture. Therefore, the client and the server application were first developed independently and later they were connected via a secured RESTful interface.

For this reason, this chapter is divided into two main sections **Server** and **Client** each explaining the development of the respective component of the project.

4.1 Security

Security of the communication is very important in this case, as the application handles sensitive data in the form of personal photographs. That is why the application should not only limit the usage of its features to authenticated users but also secure access to each specific image and record. In this case, it is going to be achieved by using *JSON Web Tokens*. JWT is an open standard that defines a compact and self-contained way for securely transmitting information between parties as a JSON object¹.

Auth0 by Okta was selected as an authentication and authorisation provider for the project. This is mainly because it provides support for both Android Jetpack Compose and Java Spring Boot and has extensive documentation.

4.2 Project Management

For this project, Git was used as a source code management tool. All source code for the application is available via the GitLab platform at the address provided in the appendix B.1.

For static code analysis, I used the *SonarLint*² extension which connects to the *SonarQube* server and is available in both IntelliJ IDEA and Android Studio.

4.3 Server

4.3.1 Languages and Technologies

In the implementation of the server were mainly used two programming languages:

■ Python 3.10.4:

¹ <https://jwt.io/introduction>

² <https://docs.sonarsource.com/sonarcloud/improving/sonarlint/>

The provided CNN algorithm is written mainly in Python. As a part of this project, the original code was slightly adjusted to provide better compatibility with the server application. The code written in Python is being run from the Java code as an external process. A more recent version of Python cannot be used because of compatibility issues with some external libraries used in the provided code.

■ **Java 21:**

The server part of the application is written in Java 21. Java is a popular object-oriented language and is also widely compatible with Kotlin.¹ Java 21 is currently the most recent version with long time support (also known as *LTS*).

Technologies that were used for the development of the server application are:

■ **Operating System:** The server application was developed and run on a device, running on *Windows 11*.

■ **Development Environment:**

For modifying the Python algorithm, it was initially used *VisualStudio Code*. The Java application was mostly developed in *IntelliJ IDEA* from JetBrains.

■ **Server Framework:**

The server-side application is developed using *Spring Boot*, a Java framework for building Spring applications. Additionally, it provides support for data management and helps deal with security issues.

■ **Build Automation Tool:**

Maven was used for dependency management and build automation of the server part of the project as it is optimised for Java applications.

■ 4.3.2 Project Structure

The server is composed of a single module that contains the Java source code as well as the Python algorithm source code. The Java application follows the **Controller-Service-Repository** pattern. If implemented correctly, this pattern aims to separate responsibilities between the presented layers to create a comprehensive and safe code [24]. Each of the layers can be found in a respective package. In addition to the Controllers, Services, and Repositories, the server application also contains a data model, data transfer objects (also called DTOs) with corresponding mappers, security configuration (as is described in sections 4.3.3 and 4.3.4), custom exception definitions, and other utils regarding image management and Python script output readings.

■ 4.3.3 Authentication

Identity authentication is the process of verifying the identity of a user or service. Based on this information, a system then provides the user with the appropriate access [25].

For server-side authentication was used the following dependency (Listing 4.1) was used to enable the Spring boot application to work with Okta via OAuth 2.0 as suggested in the Auth0 documentation².

¹ <https://kotlinlang.org/docs/faq.html#is-kotlin-compatible-with-the-java-programming-language>

² <https://auth0.com/docs/quickstart/webapp/java-spring-boot/interactive>

```

<dependency>
  <groupId>com.okta.spring</groupId>
  <artifactId>okta-spring-boot-starter</artifactId>
  <version>3.0.5</version>
</dependency>

```

Listing 4.1. Okta dependency to enable Auth0 connection on the server application

The security chain was set up as can be seen in Listing 4.2, which enables unauthorised access to health check and documentation endpoints. General access to all user data-related endpoints is restricted to authenticated users. Any requests out of the predefined addresses are automatically denied to prevent accidental data leaks.

```

@Bean
public SecurityFilterChain filterChain(HttpSecurity http)
throws Exception {
    return http.authorizeHttpRequests(it -> it
        .requestMatchers(
            SecurityURLs.PERMIT_ALL_URLS)
        .permitAll()
        .requestMatchers(
            SecurityURLs.PROTECTED_URLS)
        .authenticated()
        .anyRequest()
        .denyAll())
        .httpBasic(withDefaults())
        .oauth2ResourceServer(oauth2 ->
            oauth2.jwt(withDefaults()))
        .cors(withDefaults())
        .build();
}

```

Listing 4.2. Customised Filter Chain definition for server API

The list of protected URLs is currently defined in Listing 4.3.

```

static final RequestMatcher PROTECTED_URLS =
    new OrRequestMatcher(
        new AntPathRequestMatcher("/mapping/**"),
        new AntPathRequestMatcher("/photo/**"),
        new AntPathRequestMatcher("/record/**")
    );

```

Listing 4.3. Definition of the list of protected URLs

4.3.4 Authorisation

Authorisation is the security process that determines a user or service's level of access. In technology, we use authorisation to give users or services permission to access some data or perform a particular action [25].

Potentially sensitive images might be handled while using the application. For this reason, access to each mapping, record and image needs to be au-

thorised to prevent data leaks. For this reason, a service to provide pre-authorisation for GET requests with an ID parameter was created.

As the PhotoWrapper and Mapping entity directly store the name of the owner, the access to those types of objects can be simply authorised as shown in the `authorizePhotoAccess` (Listing 4.4) and `authorizeMappingAccess` (Listing 4.5).

```
public boolean authorizePhotoAccess(String photoId)
    throws EntityNotFoundException, InvalidIdException {
    PhotoWrapper photoWrapper =
        photoService.getPhotoWrapperById(photoId);
    return photoWrapper.getUsername().equals(getUsername());
}
```

Listing 4.4. Method defined in `AuthorizationService` for access to photographs authorisation

```
public boolean authorizeMappingAccess(String mappingId)
    throws EntityNotFoundException, InvalidIdException {
    Mapping m = mappingService.getById(mappingId);
    return m.getUsername().equals(getUsername());
}
```

Listing 4.5. Method defined in `AuthorizationService` for Mapping access authorisation

The `PhotoRecord` entity does not directly store the owner's name, so they need to be authorised either via the Mapping they belong to, or the `PhotoWrapper` that they access. The current implementation of the `authorizeRecordAccess` method is seen in Listing 4.6.

```
public boolean authorizeRecordAccess(String recordId)
    throws EntityNotFoundException, InvalidIdException {
    PhotoRecord r = recordService.getById(recordId);
    Optional<Mapping> m = mappingService
        .getMappingContaining(r);
    return m.map(a -> a.getUsername().equals(getUsername()))
        .orElse(authorizePhotoAccess(r.getPhotoId()));
}
```

Listing 4.6. Method defined in `AuthorizationService` for `PhotoRecord` access authorisation

These authentication methods are applied to the requests with an annotation as so:

```
@PreAuthorize(value = "@authorizationService
    .authorizePhotoAccess(\#id)")
```

Listing 4.7. Example of the usage on `preAuthorize` annotation

■ 4.3.5 REST API Definition

There are two principal approaches to creating REST interfaces code-first and design-first. A design-first approach involves creating a detailed API definition before writing any code, whereas a code-first approach involves writing code first and then documenting it after the fact [26].

While design-first is more beneficial for projects that need the API to be easily comprehensible and legible for the stakeholders, code-first is a faster and perfectly logical pick in the case of this application [26].

To modify or specify the endpoint definitions, swagger annotations such as `@Operation` or `@ApiResponse` were used in the code.

The API is composed of three main endpoints that provide access to the most significant entities: mapping, record and photo.

■ /mapping

- GET – Returns an overview (*data transfer object* without the list of records) of all mappings owned by the logged-in user.
- GET `/mappingId`, parameters: `boolean withDisconnected` – Retrieves complete mapping (DTO with the list of records) by the id if the mapping belongs to the logged-in user. Depending on the `withDisconnected` value, records marked as disconnected might be included.
- GET `/newest` – Returns a complete mapping marked as `newest`, if it exists. If no such mapping exists for the user, it returns `null`.
- POST `/new` – Creates a new mapping marked as `newest` if no mapping marked as `newest` exists for the user, or if the mapping currently marked as `newest` contains any records. Returns the `String` id of the mapping marked as `newest`.

■ /record

- GET `/recordId/related` – Retrieve the records marked as related to the selected one. This operation is only possible if the logged-in user is the owner of the selected record.
- PUT `/recordId/rerun` – Rerunning the AI analysis on the selected record, if the record belongs to the logged-in user.
- POST – Creating a new single record from an existing image.
- POST `/recordId` – Creating a new record based on an existing record if the existing record belongs to the logged-in user. This operation marks the two records as related.
- POST `/batch` – Creating new mapping with multiple new records that are all related to an existing record. Each of these pairs of records is marked as related.

■ /photo

- GET `/photoId` – Retrieves a compressed Base64 encoded String of an compressed image.
- GET `/photoId/result` – Retrieves a compressed result image, if it exists. If no such an image exists, the response status is 204 (meaning no content).
- POST – Saving a new image passed in Base64 encoded string with image quality check while returning the String ID of the newly created image

wrapper. If image quality is not sufficient the image is not saved and an empty String is returned.

4.3.6 Data Storage

As explained in section 3.3.1 this project is using a NoSql database. As mentioned in section 4.3.1 this also is a Spring boot application and Spring boot provides limited support for NoSql databases. According to the official documentation [27], Spring boot includes repository support for *MongoDB*, *Neo4j*, *Cassandra*, and *Couchbase*.

I decided to choose **MongoDB** as it is one of the most popular options. It also provides various alternatives regarding storing images as binary large objects, usually referred to as *BLOBs*, such as images. For example, the GridFS specification provides an opportunity to store binary objects, that are bigger than 16MB.¹ This technology can become useful if the application is used on smartphones with cameras with higher resolution.²

In MongoDB, there are three approaches on how to represent relations between objects.

The first is a database reference. String data provides an annotation `@DBRef` that can be used to automatically link and retrieve related objects. In the database, the references are stored as `DBRef('_class', 'id')`. This approach is used with classes `Mapping` and `RecordHistory` to reference `PhotoRecord`, and in `Specification` class to refer to respective `PhotoWrapper`. Database documents using this approach can be seen in Figure 4.1.

```

_id: ObjectId('662a7bb0c5aaec1ecfc17f19')
records: Array (10)
  0: DBRef('photoRecord', '662a7bb0c5aaec1ecfc17f12')
  1: DBRef('photoRecord', '662a8bccbc1411419cfbc6a0')
  2: DBRef('photoRecord', '662a8cc9bc1411419cfbc853')
  3: DBRef('photoRecord', '662bd8e914e2b911dbb25efa')
  4: DBRef('photoRecord', '662f50c89a5e4e661c2b9104')
  5: DBRef('photoRecord', '662f54949a5e4e661c2b9153')
  6: DBRef('photoRecord', '66334d3a6e3c1e5650965e24')
  7: DBRef('photoRecord', '663397f5a90dcf0ae49317b')
  8: DBRef('photoRecord', '6633dd2ca51cf65db4e2b1f2')
  9: DBRef('photoRecord', '6634d430a51cf65db4e2c471')
newest: true
username: "test@mail.com"
mappingNumber: 1
_class: "cz.cvut.fel.lemakter_bp.model.Mapping"

_id: ObjectId('662d2fcb715447dc7ed5573')
current: ObjectId('6633b4fd2d88164207eb2425')
records: Array (3)
  0: DBRef('photoRecord', '662d2bddb715447dc7ed1245')
  1: DBRef('photoRecord', '662d2fcb715447dc7ed5551')
  2: DBRef('photoRecord', '662d30a4b715447dc7ed5c7f')
_class: "cz.cvut.fel.lemakter_bp.model.RecordHistory"

```

Figure 4.1. Example of using database reference in storing instances of class `Mapping` (left) and in `RecordHistory` (right)

The second possibility is to manually reference the ID of an object. In this project, this option is used to connect a `PhotoRecord` entity with the respective `PhotoWrapper`. This approach was chosen here mainly because the contents of the wrapper are hardly accessed on the server with relation to the record and from the client side they are retrieved separately from the actual entity.

The last option that will be mentioned here is the opportunity to nest entities. This approach is used in the `PhotoRecord` entity to store `Specifications`. The reason for this is that the `Specification` objects do not have any significance unless they are a part of the record. There is also no use for these objects to be accessed or referenced from outside the related `PhotoRecord` instance.

¹ <https://www.mongodb.com/docs/manual/core/gridfs/>

² The device that was used for testing this application did not produce images over 4MB.

```

_id: ObjectId('662d2bd9b715447dc7ed1241')
photoId: "662d2bd8b715447dc7ed1240"
▼ specifications: Array (4)
  ▼ 0: Object
    _id: ObjectId('662d2bfd715447dc7ed13ed')
    croppedImg: DBRef('photoWrapper', '662d2bfd715447dc7ed13ec')
    result: "OK"
  ▼ 1: Object
    _id: ObjectId('662d2bfd715447dc7ed13ef')
    croppedImg: DBRef('photoWrapper', '662d2bfd715447dc7ed13ee')
    result: "OK"
  ▼ 2: Object
    _id: ObjectId('662d2bfd715447dc7ed13f1')
    croppedImg: DBRef('photoWrapper', '662d2bfd715447dc7ed13f0')
    result: "OK"
  ▼ 3: Object
    _id: ObjectId('662d2bfd715447dc7ed13f3')
    croppedImg: DBRef('photoWrapper', '662d2bfd715447dc7ed13f2')
    result: "UNRECOGNIZED"
placement: "LEFT_LOWER_ARM"
description: ""
createdOn: 2024-04-27T16:46:17.505+00:00
isDiscontinued: false
_class: "cz.cvut.fel.lemakter.bp.model.PhotoRecord"

```

Figure 4.2. Example of how different methods of referencing objects can be used.

All the mentioned approaches can be combined based on the use cases for different circumstances. The usage of multiple referencing options in a single entity can be seen in Figure 4.2.

The users' photographs are stored as instances of the PhotoWrapper class. This class can hold the original image, a compressed version of the image (this is the one being retrieved by client devices), and a result image (as defined in section 2.2.2). However, as can be seen in Figure 4.3 the PhotoWrappers also store the image cut-outs (as defined in section 2.2.2). These only contain the original image cut-out as any of the other image variants are not necessary. This demonstrates another benefit of storing the user data as a document instead of a table.

```

_id: ObjectId('66407bc9e57a58368743b8f9')
image: Binary.createFromBase64('/9j/4AAQSkZJRgABAQAAQABAAD/2wBDAAIBAQEBAQIBAQEAgICAgQDAgICAgUEBAMEBjUG...
username: "test"
_class: "cz.cvut.fel.lemakter.bp.model.PhotoWrapper"

_id: ObjectId('66407c0ce57a58368743b8fc')
image: Binary.createFromBase64('/9j/4AAQSkZJRgABAgAAQABAAD/2wBDAAAGBgcGBQgHBwcJCQgKDBQNDAsLDBkSEw8UHRof...
username: "test"
compressed: Binary.createFromBase64('/9j/4AAQSkZJRgABAgAAQABAAD/2wBDAAAGBgcGBQgHBwcJCQgKDBQNDAsLDBkSEw8...
result: Binary.createFromBase64('/9j/4AAQSkZJRgABAgAAQABAAD/2wBDAAAGBgcGBQgHBwcJCQgKDBQNDAsLDBkSEw8UHRo...
_class: "cz.cvut.fel.lemakter.bp.model.PhotoWrapper"

```

Figure 4.3. Example of two instances of the same class with different attributes

4.4 Client

4.4.1 Language and Technologies

The client application is written in **Kotlin 1.9.20**. The main reason is the usage of Jetpack Compose which is not compatible with any other programming languages¹. Version 1.9.20 was the most recent as of the beginning of the development of this application.

¹ <https://developer.android.com/develop/ui/compose/kotlin>


```
implementation("com.auth0.android:auth0:2.10.2")
implementation("com.auth0.android:jwtdecode:2.0.2")
```

Listing 4.8. Auth0 dependencies for working with the universal login page

```
class AuthInterceptor(private val getToken: () -> String) :
    Interceptor {
    override fun intercept(chain: Interceptor.Chain): Response {
        val requestBuilder = chain.request().newBuilder()

        requestBuilder.addHeader("Authorization",
            "Bearer \${getToken.invoke()}")

        return chain.proceed(requestBuilder.build())
    }
}
```

Listing 4.9. Definition of the `AuthInterceptor` class

To authenticate and authorise the request to the server (as described in sections 4.3.3 and 4.3.4) a request interceptor was added to a *Authorization* header with the *Bearer* token (as defined in Listing 4.9).

4.4.4 User Interface Definition

As mentioned in section 4.4.1 the framework *Jetpack Compose*¹ was used to implement the UI for the Android application. Jetpack Compose uses composable functions to define the UI programmatically.

While defining the UI, I mostly followed the Lo-Fi design and navigation, explained in section 3.3.2, using components from the *Material 3* design system².

The **My body** screen (original design in Figure 3.3) is implemented as a *LazyColumn* of expandable bars corresponding to the values of the *Placement* enum (only values for which exists at least one record are displayed). When expanded, a list of custom *RecordPreview* components is displayed. (This can be seen in the leftmost screen of Figure 4.4.) The *FloatingActionButton* in the bottom right corner of the main screen opens a *ModalBottomSheet* with buttons (as seen on the rightmost screen of Figure 4.4). The button that enables the start of a new mapping is only visible if there is at least one existing record in the current mapping.

To extend the original design (Figure 3.6) I added a *Switch* for switching between the original image and the resulting image. The reason for this extension is mentioned in section 2.2.2. (The use of this switch is demonstrated in Figures 4.5 and 4.6.) I also decided to expand the buttons and text strings on this screen to make the application more accessible.

The related records are implemented as a *LazyColumn* of custom *RelatedRecordsItem* components which are composed of an *Image* component and custom *DefText* lines, that are components, defined in the `TextUtils` class and used all around the project, to provide a simple way to manipulate the text size

¹ <https://developer.android.com/develop/ui/compose>

² <https://m3.material.io/>

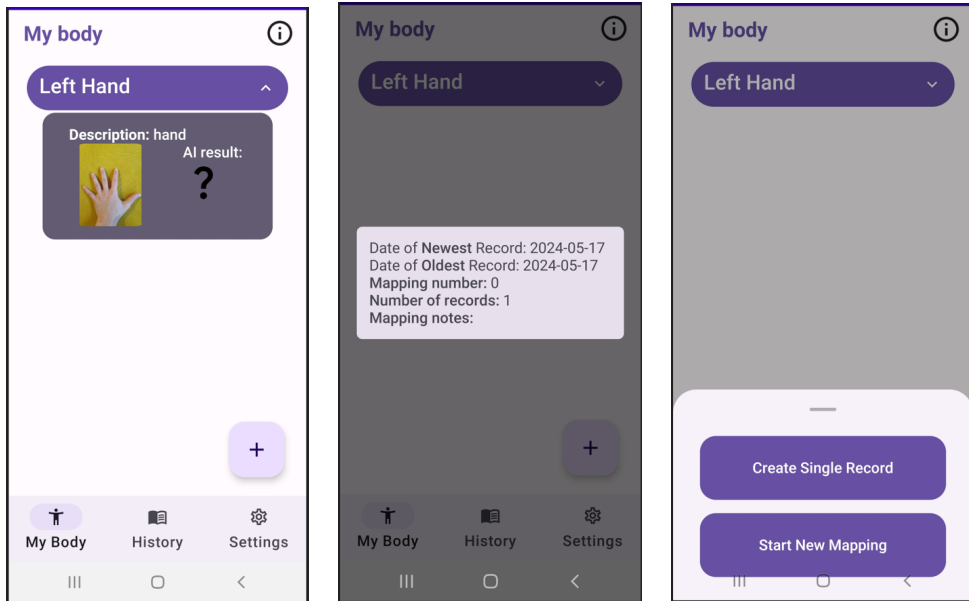


Figure 4.4. Screenshots of the *My body* screen with expanded placement bar (left), dialogue with detailed information about the mapping (middle), and expanded bottom sheet (right)

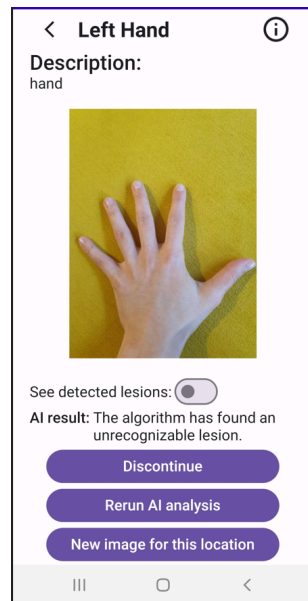


Figure 4.5. Screenshot of the record overview screen

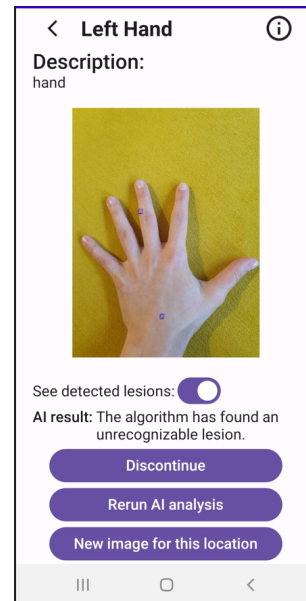


Figure 4.6. Screenshot of the record overview with detected lesions

and other attributes all around the UI. The list items are divided by *HorizontalDividers*. (As can be seen in Figure 4.8). Related records can be accessed via the button in the information dialogue of a record as is shown in Figure 4.7.

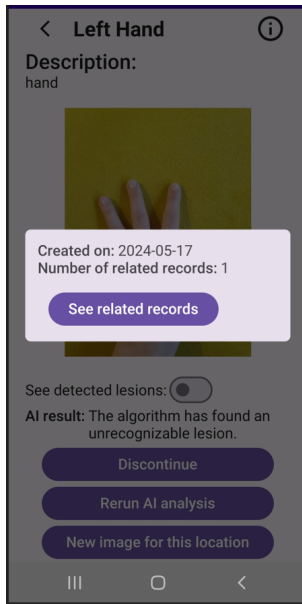


Figure 4.7. Screenshot of the dialogue with detailed information about the record



Figure 4.8. Screenshot of the list of the photos related to the selected record

The only part of the application that is vastly different from the original design is the login page. The reason for these changes was that the application is now using a universal login page as explained in 4.4.3

Therefore, a start screen (in Figure 4.9) was created to inform the user and get their consent with a button to redirect the user to the slightly customized universal login page (in Figure 4.10).

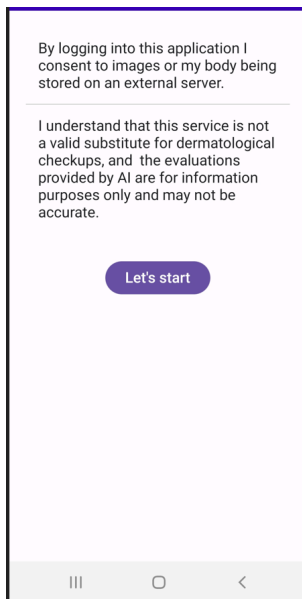


Figure 4.9. Screenshot of the final version of the starting screen

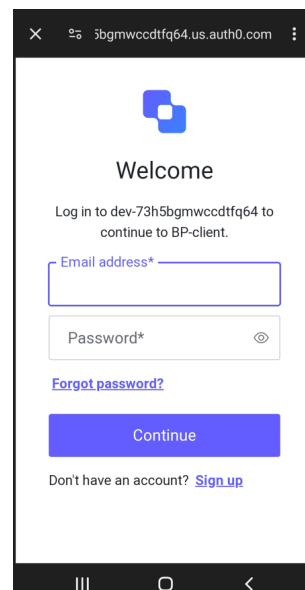


Figure 4.10. Screenshot of the customised universal login page

Chapter 5

Testing

5.1 Usability Testing

Usability testing is the practice of testing the intuitiveness of the application on a group of potential users.

In this case, a qualitative testing was conducted. Qualitative testing is usually performed on a smaller group of participants and includes close observation of the users' actions and expressions.

The target audience for this application is very wide as it is aimed to be used by people of all different ages and backgrounds. Therefore, the participants were from different age groups (ages from 12 to 53 years old) and technology comprehension skills.

Six testing sessions were conducted. Each participant was initially handed a printed copy of the user manual (appendix B.2) and asked to read through it intently first. The participants were provided with a smartphone with the application and a list of objectives to be achieved. They were also notified, that the UI was only a Lo-Fi prototype and some features were not visible, and encouraged to ask questions and speak their mind. After the objectives were accomplished the users were asked to fill out a questionnaire.

5.1.1 Questionnaire

Note that all of my responders were native Czech speakers so they were encouraged to fill out the questionnaire in Czech if that was the language they could express themselves the best. The following answers were manually translated into English.

While I am trying to relay the provided information and opinions as closely as possible, I do not guarantee that some changes to the original intent did not happen unintentionally.

If a multiplicity of the answer is greater than 1, it is denoted behind the answer as so: "possible answer – $5x$ ".

1. Age:

- 53
- 22
- 50
- 12
- 23
- 18

2. Have you ever used any application for skin monitoring and mapping?

- No – $6x$

- cannot determine
11. **How useful do you think this application is?** (*1 – very useful, 5 – completely useless*)
 - 1 – 3x
 - 2 – 2x
 - 3
 12. **Would you recommend this application to your friends/family?**
 - Yes – 5x
 - I don't know
 13. **Do you have some other notes or questions regarding the use and features of this application?**
 - When skipping a record the original one disappears which I find unintuitive, and clicking the **check image quality** uploads the record immediately without any confirmation message or something.
 - I could not find the show-related lesions button
 - Very nice and useful idea.

■ 5.1.2 Observations

This section provides a list of important observations from the testing and suggests ways how to improve these potential issues.

1. **Log In** – generally perceived as simple and comprehensible
 - Issue:** Most of the users did not see the Sign-up button at first.
 - Severity:** Minor
 - Possible solution:* The application uses a universal login page from a third-party provider. The page is customisable to a certain extent. It was decided to increase the font size and underline the link to help identify the button, but this solution has yet to be tested.
2. **Creating a Single Record** – generally perceived as quite intuitive
 - Issue:** Application does not indicate background processes.
 - Severity:** Moderate
 - Possible solution:* Adding process indicator¹ when running the image quality analysis and possibly a snackbar² when a record is successfully created could solve this issue.
3. **Discontinue a record**
 - Issue:** There is no indication that the discontinuation was successful.
 - Severity:** Moderate
 - Possible solution:* Removing the button for discontinuation for records that are already marked as discontinued (as of now, this information is only stored on the server) and adding a snackbar that the process was successful should solve this problem.
4. **Displaying related records**
 - Issue 1:** The button to display related records is hard to find.
 - Severity:** Moderate

¹ <https://m2.material.io/components/progress-indicators>

² <https://m2.material.io/components/snackbars>

Possible solution: Show the way to navigate to the button clearly in the User manual/ application Demo.

Issue 2: When showing the related records, currently only images are displayed.

Severity: Suggestion

Possible solution: Adding a possibility to click on the image to show the full overview of the related record.

5.1.3 Summary

In general, the users responded well to the application. Most of the presented problems and questions were caused by the testers' issues with the English language. Other observations mostly regarded the detection algorithm and UI. These were taken into account and compiled for future processing.

After these additions are processed, new testing needs to be conducted to see how these changes affect the accessibility and usability of the application.

5.2 Development Testing

Crucial parts of both the server and client application are covered by unit tests. Unit tests are a type of software test that focuses on testing an individual unit of the program. In this context, the units can be for example functions, methods, or objects [29].

Both, server and client projects use the *JUnit*¹ framework to manage the testing. On the server side, these are complemented with the Spring Boot annotations `@SpringBootTest` or `@DataMongoTest`. Additionally, in some cases, it was beneficial to use the *Mockito*² framework to create mock objects.

The `AuthenticationService` is a good example of a crucial element of the server that needs to be controlled by tests. The presence of the authentication token is simulated by mocking the `TokenExporter` class in the `AuthorizationService` instance as can be seen in Listing 5.1.

```
@BeforeEach
void setUp() {
    if (authorizationService == null) {
        TokenExporter tokenExporter =
            mock(TokenExporter.class);
        when(tokenExporter.getUsername(any()))
            .thenReturn(username);
        authorizationService = new AuthorizationServiceImpl(
            recordService, photoService,
            mappingService, tokenExporter);
    }
}
```

Listing 5.1. Example of the usage of Mockito framework in testing *AuthorizationService*

¹ <https://junit.org/junit4/>

² <https://site.mockito.org/>


```
@Test
void mappingAuthorizationSuccessfulTest() throws
    EntityNotFoundException, InvalidIdException {
    String id = mappingService.createNewMapping(username);

    boolean isAuth = authorizationService
        .authorizeMappingAccess(id);

    assertTrue(isAuth);

    mappingRepository.deleteById(new ObjectId(id));
}

@Test
void mappingAuthorizationFailedTest() throws
    EntityNotFoundException, InvalidIdException {
    String username1 = "test1";
    String id = mappingService.createNewMapping(username1);

    boolean isAuth = authorizationService
        .authorizeMappingAccess(id);

    assertFalse(isAuth);

    mappingRepository.deleteById(new ObjectId(id));
}
```

Listing 5.2. Example of tests for authorisation accuracy for *Mapping* access.

An example of how the accuracy of the authorisation methods can be tested is in Listings 5.2.

In the early stages of development, before the authentication was implemented, the RestAssured¹ library was used to test the functionality of server endpoints regarding image saving and retrieving. These test methods were later disabled (but not discarded) as the calls no longer work without the token.

¹ <https://rest-assured.io/>

Chapter 6

Conclusion

The final chapter will summarise the achievements of the thesis and compare them with the original assignment. This chapter will also suggest possible future extensions and improvements for the developed application.

6.1 Results

The primary objective of this thesis was to create a simple and accessible tool for secondary prevention of skin cancer in the form of a native Android application. To achieve this, I created a detailed analysis of the benefits and requirements of self-examination of skin lesions, the algorithm from [1], and techniques of photography for the best results of the detection and classification algorithm.

I designed and implemented a client and a server application, where the client is represented by an Android application with a comprehensive UI written in Jetpack Compose and the server is a Java Spring Boot application which facilitates the data storage and evaluation of the images via the AI algorithm.

A usability testing was conducted and the application mostly received positive feedback from the participants. I worked with users to compile a list of issues for future processing.

6.2 Possible extensions

This application was written with the possibility of future extensions in mind. In case of future work, the development should start by addressing the issues identified during testing and mentioned in section 5.1.2.

Following these mostly minor changes there are a lot of options as to which part of the application can be extended.

The first option is to improve the Android client. For example, the application currently does not handle creating records while offline, which is an important feature that would enhance the usability of the application. Furthermore, the current UI is based on the Lo-Fi design described in the section 3.4.1. Even though the testers mostly liked the current version, there are many possible improvements to enhance the user experience such as replacing the verbal list on the *My body* screen with a more visual representation of the human body to make the service less language dependent. As a part of this extension, an application demo can be also implemented to eliminate the need for the user manual.

Based on the usability testing, another thing that could use some additional attention is the detection part of the AI algorithm from [1]. As a part of this improvement, a feature could be added which would enable the user to mark incorrectly detected lesions (*false positives*) as well as undetected lesions to further improve the AI accuracy.

Another possible extension that has been discussed is an option to create another client application (not necessarily a smartphone one) that would be targeted towards medical specialists. It would enable the users of the existing application to connect with a doctor via the application to rule out incorrectly classified lesions and suggest treatment. Some examples of this approach are mentioned in section 2.1.3.

Lastly, I would like to mention the possibility of extending the application to the iOS platform to make the provided service accessible to a wider audience of users as together these two operating systems are used by over 99 % of all smartphone devices [4].

References

- [1] Bc. Samuel Šúr. *Detection, Classification and Matching of Skin Lesions, CTU FEE, bachelors thesis*. 2023.
<https://dspace.cvut.cz/handle/10467/109023>.
- [2] Melina Arnold, Deependra Singh, Mathieu Laversanne, Jerome Vignat, Salvatore Vaccarella, Filip Meheus, Anne E. Cust, Esther de Vries, David C. Whiteman, and Freddie Bray. Global burden of cutaneous melanoma in 2020 and projections to 2040. *JAMA Dermatology*. 2022, 158 (5), 495. DOI 10.1001/jamadermatol.2022.0160.
- [3] GSMA Press Office pressoffice@gsma.com. *Smartphone owners are now the global majority, new GSMA report reveals*. 2023.
<https://www.gsma.com/newsroom/press-release/smartphone-owners-are-now-the-global-majority-new-gsma-report-reveals/>. Accessed on 15.04.2024.
- [4] Ahmed Sherif. *Mobile OS market share worldwide 2009-2024*. 2024.
<https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/>. Accessed on 20.05.2024.
- [5] *Health promotion and disease prevention through population-based interventions, including action to address social determinants and health inequity*.
<https://www.emro.who.int/about-who/public-health-functions/health-promotion-disease-prevention.html>. Accessed on 28.04.2024.
- [6] Alexis Arasu, Nekma Meah, and Rodney Sinclair. Skin checks in primary care. *Australian Journal of General Practice*. 2019, 48 (9), 614–619. DOI 10.31128/ajgp-03-19-4887.
- [7] Valéria Kohánka, and Ferenc Kudász. *Work-related skin diseases*. 2014.
oshwiki.osha.europa.eu/en/themes/work-related-skin-diseases. Accessed on 28.12.2023.
- [8] J Daniel Jensen, and Boni E Elewski. The ABCDEF rule: Combining the “ABCDE rule” and the “ugly duckling sign” in an effort to improve patient self-screening examinations. *J. Clin. Aesthet. Dermatol.*. 2015, 8 (2), 15.
- [9] Franz Nachbar, Wilhelm Stolz, Tanja Merkle, Armand B Cognetta, Thomas Vogt, Michael Landthaler, Peter Bilek, Otto Braun-Falco, and Gerd Plewig. The ABCD rule of dermatoscopy. *Journal of American Academy of Dermatology*. 1994, 30 (4), 551–559. DOI 10.1016/S0190-9622(94)70061-3.
- [10] C Alonso-Belmonte, T Montero-Vilchez, S Arias-Santiago, and A Buendía-Eisman. [translated article] current state of skin cancer prevention: A systematic review. *Actas Dermo-Sifiliográficas*. 2022, 113 (8), DOI 10.1016/j.ad.2022.04.018.

- [11] Kushashwa Ravi Shrimali. *Image quality assessment: Brisque*. 2021. <https://learnopencv.com/image-quality-assessment-brisque/>. Accessed on 15.04.2024.
- [12] Jeffrey P Callen, David R Bickers, and Ronald L Moy. Actinic keratoses. *Journal of the American Academy of Dermatology*. 1997, 36 (4), 650–653. DOI 10.1016/s0190-9622(97)70265-2.
- [13] Valquiria Pessoa Chinem, and Hélio Amante Miot. Epidemiologia do carcinoma basocelular. *Anais Brasileiros de Dermatologia*. 2011, 86 (2), 292–305. DOI 10.1590/s0365-05962011000200013.
- [14] William E Damsky, and Marcus Bosenberg. Melanocytic Nevi and melanoma: Unraveling a complex relationship. *Oncogene*. 2017, 36 (42), 5771–5792. DOI 10.1038/onc.2017.189.
- [15] M B Morgan, Gary L Stevens, and Stephen Switlyk. Benign lichenoid keratosis. *The American Journal of Dermatopathology*. 2005, 27 (5), 387–392. DOI 10.1097/01.dad.0000175533.65486.84.
- [16] John T Mullen, Lei Feng, Yan Xing, Paul F Mansfield, Jeffrey E Gershenwald, Jeffrey E Lee, Merrick I Ross, and Janice N Cormier. Invasive squamous cell carcinoma of the skin: Defining a high-risk group. *Annals of Surgical Oncology*. 2006, 13 (7), 902–909. DOI 10.1245/aso.2006.07.022.
- [17] Joao Vítor Alves, Diogo Miguel Matos, Hugo Frederico Barreiros, and Elvira Augusta Bártolo. Variants of dermatofibroma - A histopathological study . *Anais Brasileiros de Dermatologia*. 2014, 89 (3), 472–477. DOI 10.1590/abd1806-4841.20142629.
- [18] Vincenzo Piccolo, Teresa Russo, Elvira Moscarella, Gabriella Braccaccio, Roberto Alfano, and Giuseppe Argenziano. Dermatoscopy of vascular lesions . *Dermatologic Clinics* . 2018, 36 (4), 389–395. DOI 10.1016/j.det.2018.05.006.
- [19] Rebecca Ellison. *Why I avoid the front facing “selfie” Camera for business photos; rebecca ellison creative - brand coach and photographer*. 2021. <https://rebeccaellison.com/blog/why-avoid-front-facing-selfie-camera-for-business-photos-instagram/>. Accessed on 20.04.2024.
- [20] John Terra. *What is client-server architecture? everything you should know: Simplilearn*. 2023. <https://www.simplilearn.com/what-is-client-server-architecture-article>. Accessed on 13.05.2024.
- [21] Joseph Molloy. *A comprehensive overview of the client-server model*. 2023. <https://www.liquidweb.com/blog/client-server-architecture/>. Accessed on 14.05.2024.
- [22] AltexSoft. *Unstructured data, explained*. 2023. <https://www.altexsoft.com/blog/unstructured-data/>. Accessed on 14.05.2024.
- [23] Barna Burom. *NoSQL vs relational database file storing (mongodb and SQL Server comparison)*. 2021. <https://codingsans.com/blog/nosql-vs-relational-database>. Accessed on 14.05.2024.

-
- [24] Paul Michaels. *The service / repository pattern*. 2023.
<https://pmichaels.net/service-repository-pattern/>. Accessed on 16.05.2024.
- [25] *Authentication vs. Authorization*.
<https://www.onelogin.com/learn/authentication-vs-authorization>. Accessed on 13.05.2024.
- [26] McKenzie Tucci. *Code-first vs. design-first: Eliminate friction with api exploration*. 2023.
<https://swagger.io/blog/code-first-vs-design-first-api/>. Accessed on 13.05.2024.
- [27] *Working with NoSQL Technologies :: Spring Boot; References; Data*.
<https://docs.spring.io/spring-boot/reference/data/nosql.html>. Accessed on 15.05.2024.
- [28] W. Denniss, and J. Bradley. OAuth 2.0 for Native Apps. *OAuth 2.0 for native apps*. 2017, DOI 10.17487/rfc8252.
- [29] Nickolay Bakharev. *Unit testing: Definition, examples, and critical best practices*. 2024.
<https://brightsec.com/blog/unit-testing/>. Accessed on 16.05.2024.

Appendix A

List of Abbreviations

AI	■ Artificial Intelligence
API	■ Application Programming Interface
BLOB	■ Binary Large Object
BRISQUE	■ Blind/Referenceless Image Spatial Quality Evaluator
CLI	■ Command Line Interface
CNN	■ Convolutional Neural Network
CSV	■ Comma Separated Values
DTO	■ Data Transfer Object
HTTPS	■ Hypertext Transfer Protocol Secure
IDE	■ Integrated Development Environment
JWT	■ JSON web token
Lo-Fi	■ Low-Fidelity
mp	■ megapixels
MVVM	■ Model-View-View-Model
PIQ	■ PyTorch Image Quality
REST	■ Representational State Transfer
RGB	■ Red, Green, Blue
UI	■ User Interface
UID	■ Unique Identifier
UML	■ Unified Model Language
URL	■ Uniform Resource Locator
UV	■ Ultra Violet
WHO	■ World Health Organization

Appendix B

List of Electronic Appendices

B.1 Source Code

The source code for the application is also available on <https://gitlab.fel.cvut.cz/lemakter/bakalarska-prace>. The same contents as this repository can be found in the provided sources in folder *bakalarska_prace*. Subsequently, the *server* folder contains the source code for the server-side application and the *client* folder contains the source code of the client-side application.

B.2 User Manual

The original user manual for the application is provided in the file *UserManual.pdf*.

Appendix C

Placement Photography List

List of places to photograph that was provided to the subjects in the study (section 2.3.1):

- Chest
- Stomach
- Upper back
- Lower back
- Buttocks
- Right upper arm
- Right lower arm
- Right hand
- Right thigh – front side
- Right thigh – back side
- Right calf
- Right shin
- Right foot
- Neck
- Face
- Head