Diploma thesis

Czech Technical University in Prague

Android application for acupressure

Petr Ješke *et al.*

Supervisor: Ing. Václav Burda
April 2020

Acknowledgements

Firstly I would like to thank my family for their support during my studies at the university. Furthermore, I thank my supervisor Václav Burda for his support and advice. Finally, I would like to thank Daniel Novák and Jindřich Prokop for their comments and advice.

Declaration

I declare that this work is all my own and all sources are citated in the bibliography.

Prague, April 5, 2020

Prohlašuji, že jsem předloženou práci vypracoval samostatně, a že jsem uvedl veškerou použitou literaturu.

V Praze, 5. dubna 2020

Abstract

The goal of this thesis is to implement a mobile application, which will offer its users the opportunity to deal with their health issues using acupressure. An analysis of existing solutions was made, followed by the design of the application. The application was implemented in line with the assembled design and tested with a series of user tests. The process of modern mobile application design is studied in detail together with its implementation. The thesis also contains a discussion of vital signs estimation, namely heart and respiratory rate estimation, using camera footage and implementation of this functionality in the Android environment.

Keywords: acupressure, Android, vital signs estimation using mobile phone camera

Supervisor: Ing. Václav Burda

Abstrakt

Cílem této diplomové práce je implementovat mobilní aplikaci, která nabídne jejím uživatelům možnost řešit jejich zdravotní problémy pomocí akupresury. V rámci práce byla provedena analýza existujících řešení, následovaná návrhem a implementací aplikace a zakončená řadou uživatelských testů. Práce obsahuje detailní rozbor procesu vývoje moderní mobilní aplikace spolu s její implementací. Součástí práce je také diskuse nad možnostmi estimace vitálních znaků uživatele, jmenovitě detekce tepové a dechové frekvence, za použití kamery a implementace této funkcionality v prostředí Android.

Klíčová slova: akupresura, Android, estimace vitálních znaků s použitím kamery mobilního telefonu

Překlad názvu: Android aplikace pro akupresuru

Contents

MASTER'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Ješke Petr** Personal ID number: **456893**

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

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

Study program: **Open Informatics**

Specialisation: **Software Engineering**

II. Master's thesis details

Master's thesis title in English:

Android application for acupressure

Master's thesis title in Czech:

Android aplikace pro akupresuru

Guidelines:

The goal of the thesis is to implement and test an application for Android devices, which will help its users with their health problems through acupressure.
1) Familiarize with the given problematic and analyze existing solutions
2) Create a scheme and design of the application, which will reflect current google design guidelines
3) Implement application based on the defined scheme and design
4) Try to implement breath detection using the phone camera
5) Verify the application functionality using at least five users

Bibliography / sources:

[1] Irshad, Shafaq & Awang Rambli, Dayang. (2017). Advances in Mobile Augmented Reality from User Experience Perspective: A Review of Studies. 466-477. 10.1007/978-3-319-70010-6_43.
[2] Christina Mildt – Akupresura v praxi – 2017, překlad 2.vydání, Grada Publishing, a. s.

Name and workplace of master's thesis supervisor:

Ing. Václav Burda, Analysis and Interpretation of Biomedical Data, FEE

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

Date of master's thesis assignment: **12.02.2020** Deadline for master's thesis submission: **22.05.2020**

Assignment valid until: **30.09.2021**

Ing. Václav Burda 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 master's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

Date of assignment receipt Student's signature

Chapter 1

Introduction

Acupressure is an alternative medicine technique originating from traditional Chinese medicine. Traditional Chinese medicine is based on the concept of life energy, which flows through the whole body and concentrates at special points known as meridians. Acupuncture and acupressure use the stimulation of these points to release pain and cure various diseases. While acupuncture uses needles to stimulate these points, acupressure uses physical pressure onto those points. The benefit of acupressure compared to acupuncture is, that even the layman can perform acupressure exercises and stimulate the meridians on his own, whereas acupuncture requires a professional skilled in inserting needles in the right combinations onto the meridians.

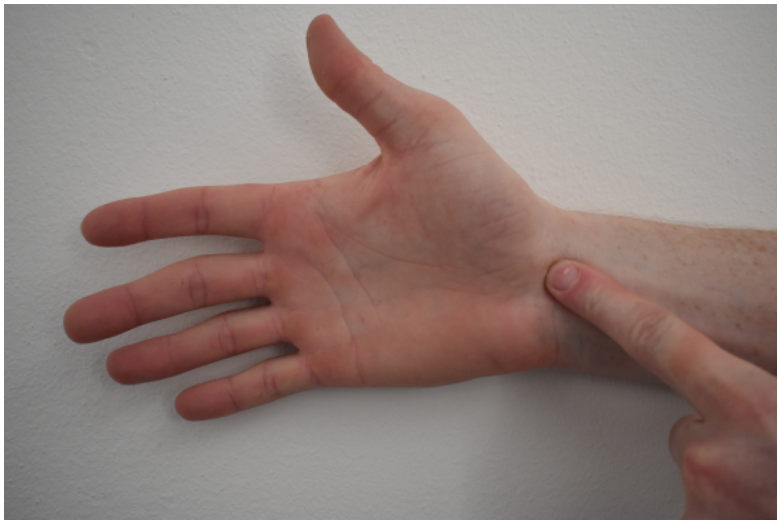


Figure 1.1: Acupressure point known as Spirit Gate used in the application

Acupressure, although considered an alternative medicine technique, is used throughout the world, but the availability of resources from which a layman might learn how to perform acupressure is low. Today, anyone wishing to start with acupressure has just two options . Search in the literature or find a professional, who will teach them how to use acupressure. The goal of this thesis is to add another option to this set, a mobile application, which

will serve as a study tool and in addition as a database of acupressure exercises.

Although books are an invaluable source of information, most of the interviewees from the initial survey would choose a mobile application as a study tool, as it comes with several benefits:

- Mobile applications are highly portable, which means that when some problem arise, the user of the application can quickly find the solution for the problem in his/her pocket.
- Compared to books, the application may contain a high number of images, and the user may zoom into the image to find an exact location of the acupressure point.
- The price of a mobile application is significantly less than any course teaching how to perform acupressure under the guidance of a professional.
- The application may also contain educational videos or 3D models, which can also facilitate the localization of acupressure points.

Chapter 2

Planned functionality analysis

The first part of this chapter contains a discussion about the best operating system for the application, followed by an analysis of the versions of the selected operating system. The second part of this chapter contains a formal definition of planned functionality in the form of functional and non-functional requirements.

2.1 Platform selection

The following table (Table 2.1) contains statistics of the smartphone market share from the past five years. As can be seen, the Android OS dominates the market, with a stable 70% share. The second most successful OS is iOS, with a 20% share. An observation can be made that the share of the remaining operating systems is decreasing rapidly. In 2015, the remaining operating systems had approximately 15% in 2019, approximately 1%. Based on this analysis, the Android OS was selected as the best platform for the application.

Year	**2015**[%]	**2016**[%]	**2017**[%]	**2018**[%]	**2019**[%]
Android	65.9	72.0	73.5	75.2	74.1
iOS	19.2	18.9	19.9	22.0	24.8
KaiOs	0.0	0.0	0.0	1.1	0.4
Windows	2.3	1.2	0.6	0.3	0.1
Other	12.6	7.9	5.9	1.4	0.6

Table 2.1: Market share of mobile platforms in the past 5 years[1].

The next phase of the development process is the selection of the supported backward compatibility, in particular, the selection of the minimal version of the Android OS required in order to install and run the application successfully. Based on the January 2020 statistics show below (Table 2.2), Lollipop was selected as the minimum supported version of the OS, which will guarantee compatibility of the application with at least 94% of devices currently running.

Name	Api Level	Usage[%]
Pie	28	41.9
Oreo	26-27	22.8
Nougat	24-25	11.8
Marshmallow	23	10.6
Lollipop	21-22	7.1
Total	21-28	94.2

Table 2.2: A relative number of devices running a given version of the Android platform[1].

2.2 Application functionality

The core functionality of the application is to provide a database of acupressure exercises to the user. For fast navigation, the application will offer logical sectioning, and filtering of acupressure points based on their location and symptoms to be resolved with the given points. Each exercise should be well described, so a user with no previous experience will be able to perform the procedure correctly. For this, each exercise will consist of a verbal description, together with a set of images describing the location of the point.

2.2.1 Functional requirements

Application tutorial

The application should contain a tutorial or a set of hints to teach the user how to use the application. The tutorial can be implemented as an onboarding tutorial, which will be triggered in the first run of the application and will not be triggered in following application startups. Another option is a set of hints available to the user in appropriate screens and which can be displayed when required.

GUI to search for points based on their location

The application should contain a tool to search for desired points based on the location of the point on the body part. The filter options for body part filtering should be as follows:

- Head
- Torso
- Upper limbs
- Lower limbs

Search tool for a specific problem solutuion

Besides the location-based search tool, the application should also provide a tool to search for points to help the user deal with a specific problem. Example: If the user suffers from headaches, the application should offer a list of acupressure points to solve this particular problem.

Automatic data synchronisation

All the application data should be automatically synchronized with the application server on a regular basis. By default, the synchronization should be allowed only via WiFi connection, but the user should also have the option to allow mobile data synchronization.

Acupressure points prioritization

The user should be able to mark frequently used points as favourites. The favourite points should be easily accessible in the application.

Well structured acupressure point description

Each point should be described with the following pieces of information:

- Localisation - verbal description of the exact location of the point
- Application - instructions on how to use the point (how much much pressure should be used during the exercise, how often the user should perform the exercise)
- Description - description of the effects of the point, how the point influences the organism
- Images - each point should be assigned an image or set of images showing the location of the point

Zoomable images

Each image showing the position of the given acupressure point, should be zoomable, so the user can easily find the exact location of the point.

Feedback mechanism

For future application improvement, the user should be able to send feedback from within the application. The application should provide the user the following feedback options:

- Reporting errors in the application
- Improvement suggestions
- Question form

Vital signs estimation

The application should contain a tool for vital signs estimation, specifically a tool which measures the heart and respiratory rate of the user. This tool should use the phone's camera for this measurement.

Results of vital signs measurements

The user should be able to list the results of the performed measurements from within the application.

2.2.2 Nonfunctional requirements

Version of the OS

The application should run on at least 90% of devices of the selected OS which are currently in use. A discussion about the versions of the OS is placed at the beginning of this chapter.

Language

The application should support at least two language sets (cs, en) and should be implemented in such a way that adding new language sets will not require extensive modifications of the application.

Design guidelines

The application should follow current Google design guidelines. It should also use contemporary application architecture.

Chapter 3

Existing solutions

This chapter contains a summary of applications dedicated to acupressure, which are available for Android users. The Google Play store[2] offers various applications for mobile acupressure, although none of the applications provide sufficient solutions for the desired functionality defined in chapter 2. Moreover, most of the applications available are unfinished or in beta version only.

3.1 Acupressure Point for 300+ Diseases

"Acupressure Point for 300+ Diseases"[3] was selected as the first example of existing solutions as it was a number one application on Google Play store at the time of writing this thesis. The application provides a very simple solution for the given problem (Figure 3.1). However, the application only offers an alphabetically ordered list of acupressure exercises. The user has no option to filter the exercises. It is, therefore, hard to find any desired exercise as the user has to scroll through the whole list. Such an approach is very inefficient. Another difficulty is that the application is incomplete. The user can enter a screen, from which there are no options to move to any other screen. Sometimes the user may even end up with a completely white screen. Information about each point is limited to a verbal description of its location. Any descriptions of how to use the point or how the point influences the organism are missing.

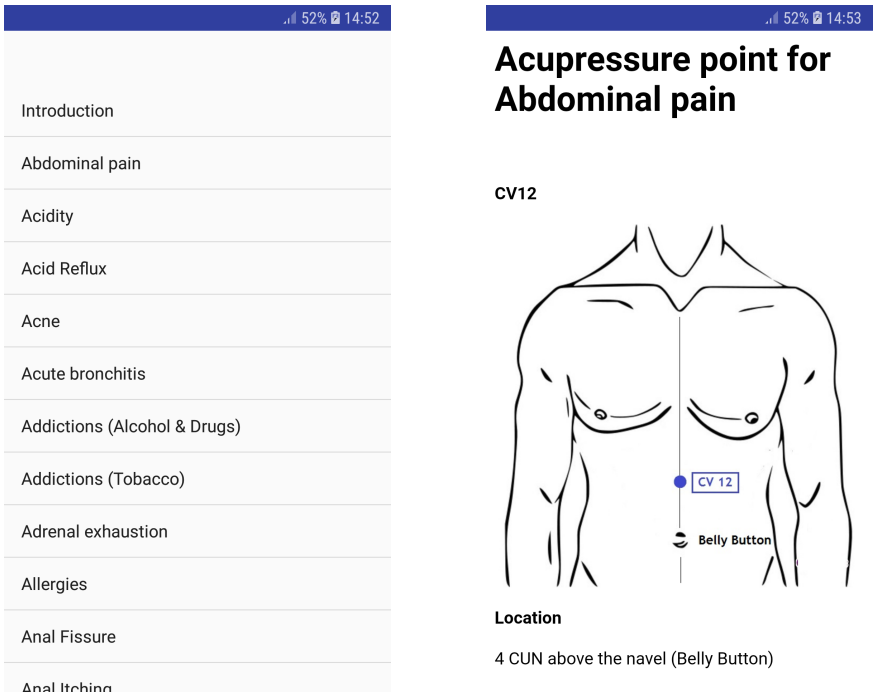
Summary

Pros:

- Wide database of exercises

Cons:

- No tool for effective exercise filtering/searching
- Unsatisfactory exercise description with low resolution images
- Broken application flow

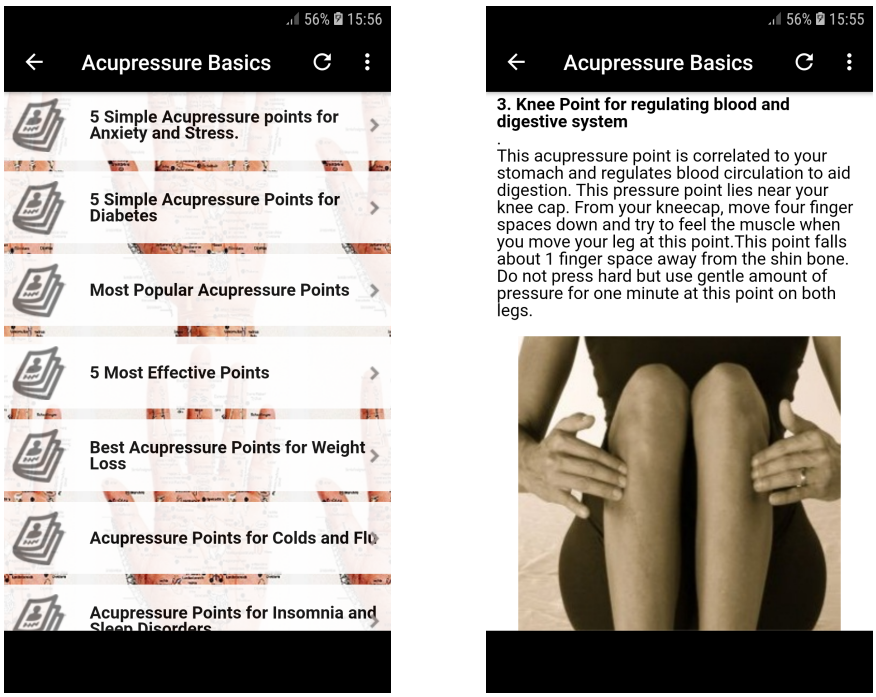


(a) : Linear list of difficulties **(b) :** Abdominal pain exercise description

Figure 3.1: The GUI of the application Acupressure Point for 300+ Diseases

3.2 Basics of Acupressure Massage

The second application selected for analysis is called "Basics of Acupressure Massage"[4]. The application is shown in the following figure 3.2. This application also lacks proper structure. The user interface is chaotic, and it is very difficult to find the desired acupressure point (the point which would solve the specific problem). The entire application is inconsistent, photos of various sizes are mixed with hand drawn pictures and images, intensifying the feeling that the application is not finished and that it is only a prototype.



(a) : Navigation screen **(b) :** Point description screen

Figure 3.2: The GUI of the application Basics of Acupressure Massage

Summary

Pros:

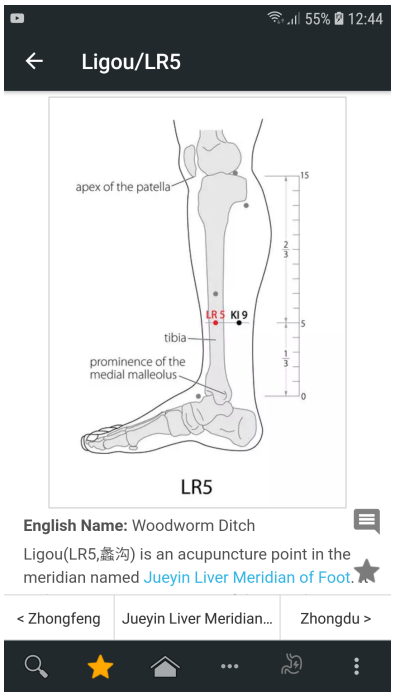
- Better, but still insufficient, point description compared to the previously studied application

Cons:

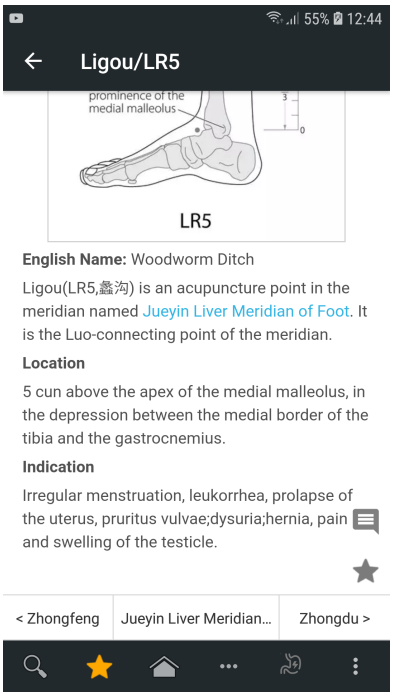
- No tool for effective point filtering
- Chaotic application structure
- Inconsistent image gallery throughout the application

3.3 Acupuncture Index

The third application used for analysis is called "Acupuncture Index" [5]. As the name suggests, the application is dedicated to acupuncture, but since both acupressure and acupuncture require similar application functionality, it was also included in the analysis. At first sight it is apparent that Acupuncture index is more sophisticated than the two previously examined applications. It offers a large database of acupuncture points. Each point description contains a well-arranged image (Figure 3.3a)) together with a location description and a summary of the impacts of the points on the organism (Figure 3.3b)).



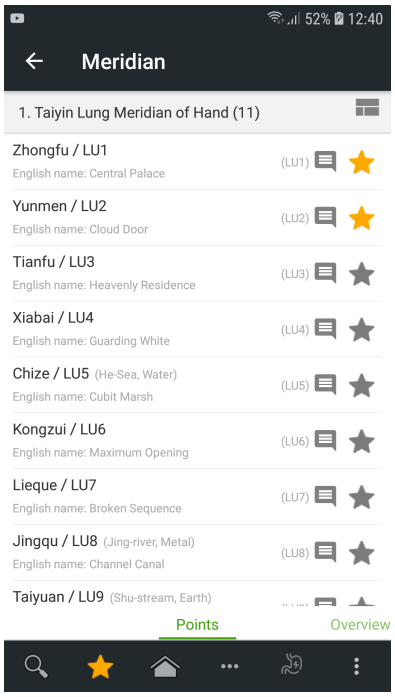
(a) : Reproduction of image describing the location of LR5 point



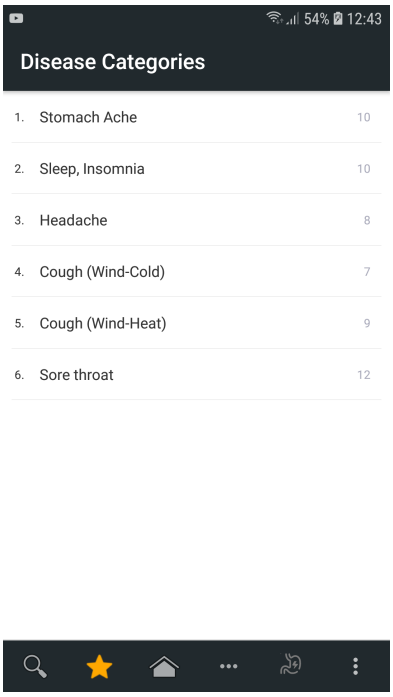
(b) : Description of the location and indications of the LR5 point

Figure 3.3: LR5 point description in the Acupuncture Index application

The application is also more explicit and better organized compared to the two previously studied applications. All points are divided into groups by the meridians[1] to which they belong (Figure 3.4a). Besides the meridian division, the application offers the user an option to create groups of points. There is also a screen that enable searching for points by specific diseases (Figure 3.4b)). However, the disease list contains only six items. The application also contains a search tool, although testing of this feature showed that it does not work very well.



(a) : Division of acupuncture points by meridians



(b) : Disease based acupuncture point division

Figure 3.4: Acupuncture index application structure demonstration

Summary

Pros:

- Large database of acupuncture points
- Consistent and well-arranged images used for point description
- Option for custom point division
- Search tool for specific points and disease-based point division

[1] a meridian in Chinese medicine represents a single line/path of points through the body

Cons:

- Search tool not fully functional
- Small number of diseases for point division
- Does not respect Google design guidelines (for example one navigation bar on top of another etc.)

3.4 Analysis Summary

As can be seen from the above examinations of existing solutions, there are no applications offering satisfactory solutions to the given problem. Even the last examined application, which could be seen as the closest equivalent for the developed application, does not fulfill the required functionality, apart from its focus on acupuncture instead of acupressure. It is, therefore, meaningful to deal with the problem of mobile acupressure and provide a solution which would offer the user a well-arranged application, with consistent GUI, which respects Google design guidelines and current application trends.

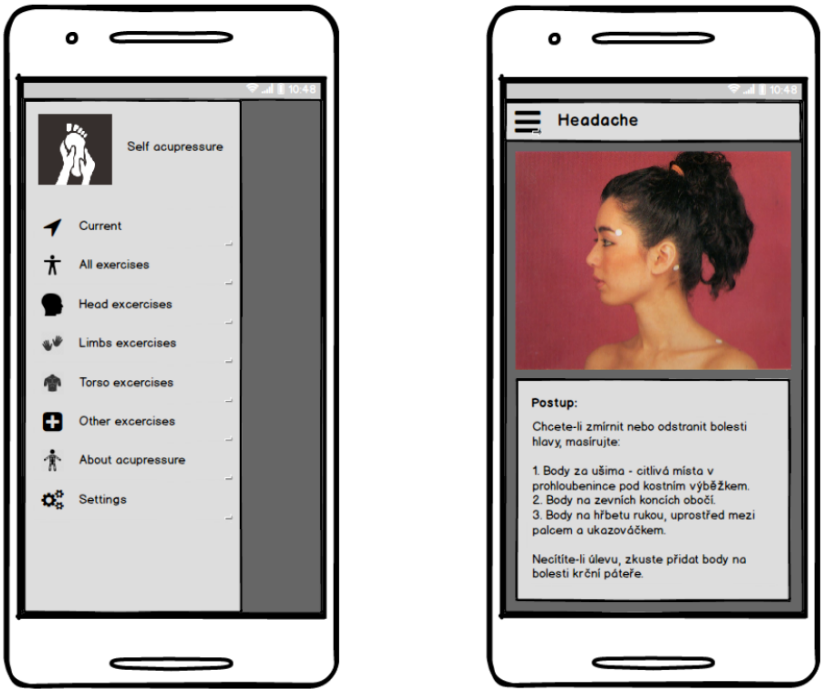
Chapter 4

Design

4.1 User interface design

4.1.1 Balsamiq Mockups

The first application concept was created using the program Balsamiq Mockups[6]. Balsamiq Mockups is a graphical user interface tool used for designing the GUI of web pages and mobile applications. The designer is able to quickly create a required design using the drag-and-drop editor with a combination of the large set of Balsamiq pre-built widgets. After several iterations, the initial design, shown below (Figure 4.1), was created. The complete mockup is placed in the attachment.



(a) : Screen with a side navigation bar **(b) :** Point detail screen

Figure 4.1: The initial mockup created in Balsamiq Mockups

4.1.2 Adobe XD

The initial design was used to fully define the functional requirements of the application. It helped to define the basics of the application structure and the data model for the application. It was also used as the basis for the second mockup created using Adobe XD[7]. Adobe XD is a new UX design tool, which can be used to create high-quality application design. The most significant advantage of Adobe XD is that it can be used to create a mockup, which looks like (and behaves like) the final application. And it is, therefore, possible to test the application flow and its structure in the early stage of the development process. The advanced mockup (Figures 4.3 - 4.6) created after several iterations reflects following scheme (Figure 4.2).

- Main screen bottom toolbar
  - Favourite screen ..... Figure C.1
    - Point detail screen ..... Figure C.6
      - Image detail screen ..... Figure C.7
  - Search by problem screen ..... Figure C.2
    - Point list screen ..... Figure C.5
      - Point detail screen ..... Figure C.6
        - Image detail screen ..... Figure C.7
  - Search by bodypart screen ..... Figure C.3
    - Point list screen ..... Figure C.5
      - Point detail screen ..... Figure C.6
        - Image detail screen ..... Figure C.7
  - Advanced scren ..... Figure C.4
    - Send feedback screen
    - Vital signs screen bottom toolbar
      - Measurement screen ..... Figure C.8
      - How to measure screen ..... Figure C.9
      - Measurement results screen ..... Figure C.10

Figure 4.2: Scheme of the initial design created in Adobe XD

The structure of the initial design was reworked. The navigation bar was added to the bottom of the screen, and the application was divided into four main screens.

- Home screen (Figure 4.3), which contains the users favourite points.
- Search by problem screen (Figure 4.4), where the user can look for points to help with particular problems.
- Search by body part screen (Figure 4.5), which replaces the side navigation bar from the initial mockup. Using this screen, the user can get acupressure points for specific body parts by simply tapping on the desired location.
- Advanced screen, which contains application settings, options to send feedback to the author, and an entry point for measuring vital signs rate.

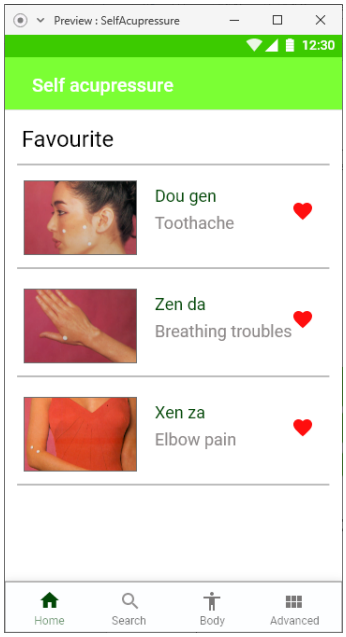


Figure 4.3: Home screen

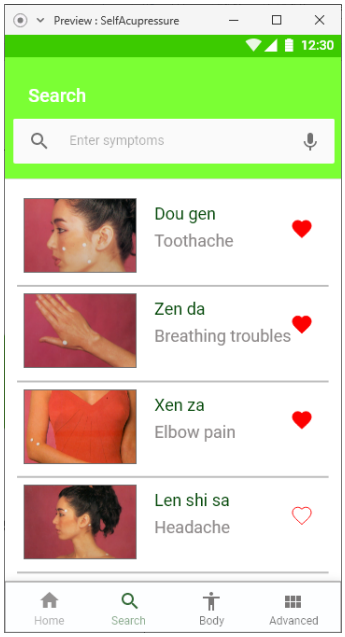


Figure 4.4: Search screen

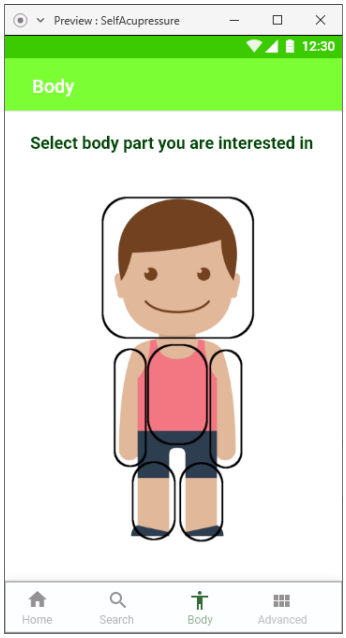


Figure 4.5: Body screen

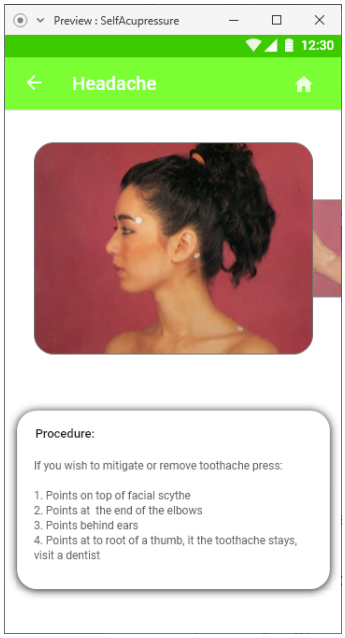


Figure 4.6: Point detail screen

4.2 Architecture desingn

After the design was completed, the next phase was selecting the architecture pattern for the application. This section describes the most common architectures for android applications. A short summary of their pros and cons is presented, together with a discussion about the best option.

4.2.1 Importance of the Android application design

Mobile applications have, unlike desktop apps, a very complex structure. Desktop applications work as a single process, rarely interacting with other programs. On the other hand, the Android application is usually a sophisticated structure of various application components such as activities, fragments, services, etc. Moreover, mobile applications often iteract with other mobile applications and can be interrupted by the behaviour of other applications at any time.

The complexity of Android applications is demonstrated with the following example. Let us say that the application SampleApp provides an option to send a message from within the application, using another app like Gmail or Messenger. The flow of the actions of the Android OS can be described as follows:

- The Sample app triggers a Messenger intent. This means that the SampleApp is put onPause(), and the Android OS runs the Messenger app to process the request.
- In the Messenger app, the user writes the content of the message and decides to add a photo as an attachment.
- This requires putting the Messenger onPause(), saving the content of the message, and triggering the Camera application.
- In the end, the user sends the message and returns back to the Sample app.

It is also possible that the application flow is interrupted by an incoming phone call, which requires another pause of the application. The OS may even terminate some applications running in the background to free resources for remaining applications.

The example provided shows the importance of the Android application architecture. The architecture has to be designed with respect to the fact that the application may be paused or even terminated at any time, which means that it is not possible to store any application data or application state in the application components.

4.3 Common application architectures

The most common application architectures (oldest first) are as follows:

- MVC (Model View Controller)
- MVP (Model View Presenter)
- MVVM (Model View ViewModel)

4.3.1 Model View Controller

The MVC is a pattern used mostly for web forms but can be sufficient for small applications. This architecture focuses on separating the user interface (UI) from the data model. For that, the MVC pattern uses three components (Figure 4.7). By default, all three components are able to interact with each other.

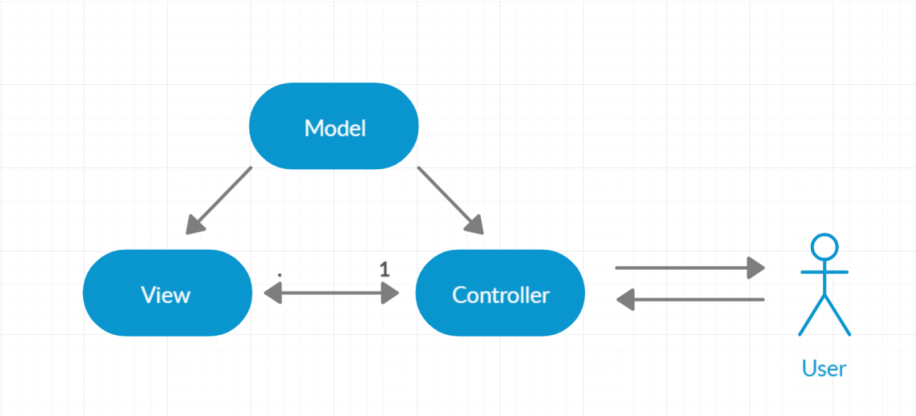


Figure 4.7: Model View Controller architecture

- The Model is responsible for business logic. It defines the rules for handling data.
- The View deals with displaying the data, which it obtains from the Controller.
- The Controller is responsible for handling UI actions (button taps, swipe gestures, etc.). General practice is to have one controller, which handles multiple views.

4.3.2 Model View Presenter

The MVP (Figure 4.8) is the successor to the MVC. It uses the Model and the View components from the previous model and replaces the Controller for a Presenter. Finally, it adds a new layer (an interface) between the View

and the Presenter. This increases the encapsulation of the application, and it allows mock Views for unit tests.

- The Model and View fulfill the same obligations as in the MVC
- The View Interface provides a loose coupling between the View and the Presenter
- The Presenter is responsible for interacting between the View and the Model. (Note that there is no direct connection between the View and the Model). Usually, the relationship between the Presenter and the View is one to one.

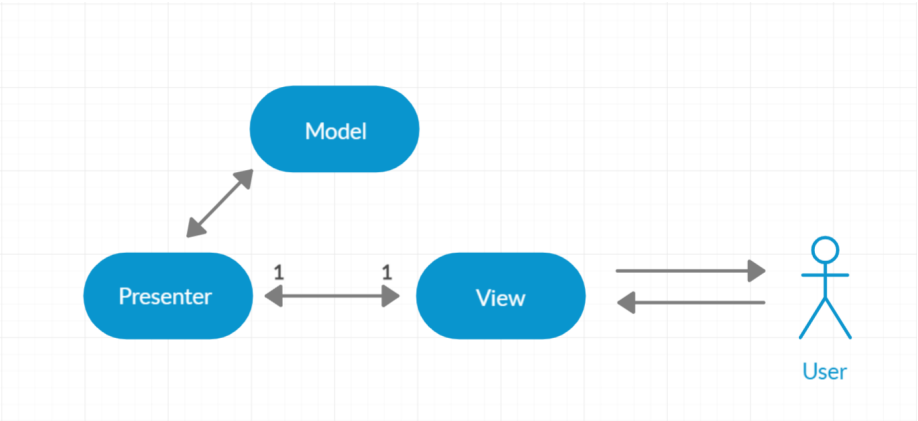


Figure 4.8: Model View Presenter architecture

4.3.3 Model View ViewModel

The MVVM (Figure 4.9) is the youngest architecture pattern in the set. The View and the Model are again the same as in the MVC pattern. In this case, the Controller was replaced by the ViewModel. The ViewModel provides the data for a specific UI component (View), but this time the ViewModel supports a two-way data binding with the View. This allows the automatic propagation of data changes between the Model and the View (the process of automatic propagation of data changes is described in the following pages).

- The View displays certain data. But it does not own/hold the data. It simply performs rendering of the data, which it obtains from the ViewModel.
- The ViewModel is designed to hold certain data objects created from the data obtained from the Model, and it is able to call functions of the View to render these data objects.
- The Model holds the data. If the ViewModel requires certain data, the Model is responsible for obtaining them (usually from the local database or from the remote host) and passing them on to the ViewModel.

The Model View ViewModel is the most advanced pattern and the best option for large applications. Thanks to the separation of individual components, the architecture is easily testable and modifiable. Based on this comparison, the MVVM pattern was selected for the application.

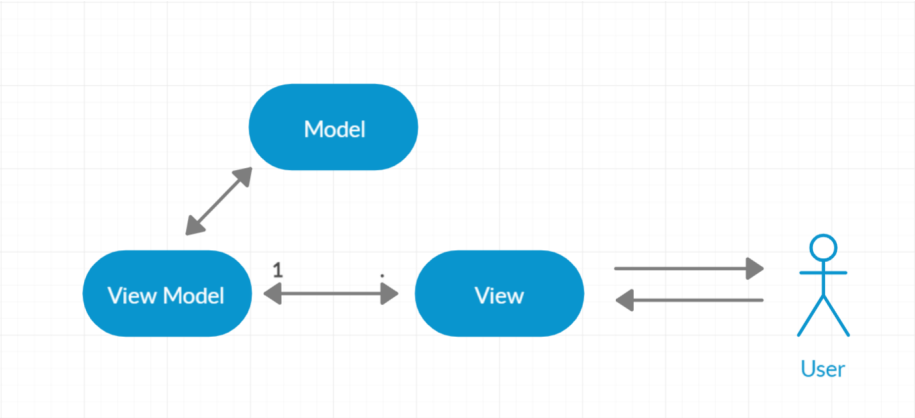


Figure 4.9: Model View ViewModel architecture

4.3.4 Model View ViewModel in Android applications

The MVVM pattern is currently the recommended option for the Android application architecture (based on official Android documentation[8]). The structure of the MVVM pattern for Android applications is shown in the figure below (Figure 4.10).

Activity/Fragment

Activities and fragments are studied in detail in section 5.2. In the MVVM pattern, activities and fragments represent the View object. They can be seen as a tool that receives data from its ViewModel and is responsible for rendering the data in a predefined way.

Repository

The Repository is the data provider for the application. By default, the Repository can be a simple java class, which provides the data, usually using a 3rd party library/tool. While each activity/fragment has its own ViewModel, the Repository may provide data for multiple ViewModels. The Repository is the only class in this model, which may depend on multiple other classes. The number of dependencies varies according to particular applications (whether the application only uses a local database or is connected to a remote data source or other source of the data). In the example provided, the Repository is connected to a local SQL database via the Room library and to a remote data source via the Retrofit Tool.

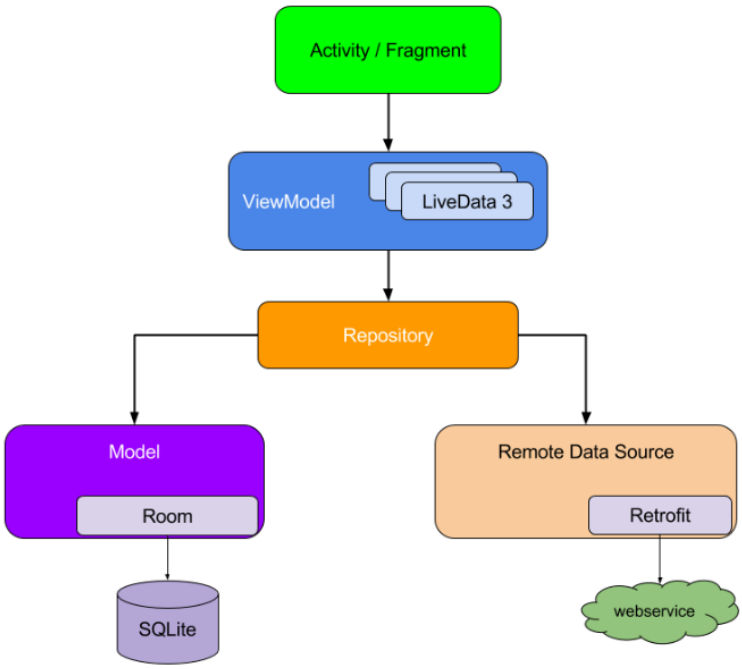


Figure 4.10: MVVM model implemented in Android [8]

ViewModel

The ViewModel establishes a connection between the activity/fragment and the Repository. Every activity/fragment which uses data that is not completely static (data which may change over time, such as database records), should be provided its ViewModel. The ViewModel is responsible for obtaining the data from the Repository. It may perform filtering or modification of the data.

The most important part of the ViewModel is a construct called LiveData, which provides a simple implementation of the observer pattern. The use of the LiveData significantly improves the communication between the ViewModel and the Repository. The principle of the observer pattern is that one object, usually called the subject, maintains a set of observers, and notifies them automatically when the state of the observed data changes. Without the observer pattern, the ViewModel would have to periodically verify if the data it works with are valid (which would require constant confirmation from the Repository that the data are current). With the observer pattern, the ViewModel is automatically notified if the LiveData changes. For example, if the ViewModel is responsible for rendering data from the server, it can initially process stored data, then concentrate on user actions, and when the Repository obtains new data from the server, the ViewModel is notified automatically to process new data.

Chapter 5

Implementation

5.1 Programing in Android in general

The Android OS, which runs on a modified version of the Linux kernel, was initially released in 2008, and has been the best-selling OS on smartphones since 2011. As shown in chapter 2, today, more than 70% of smartphone devices use the Android OS.

5.1.1 Programing languages

The initial language for the Android applications was Java. Java is an object-oriented programing language and the primary choice for most of the Android developers as the OS of Android itself is written in Java (The UI of Android is written in Java, while the core uses the C language, as it is derived from the Linux kernel.). In 2011 IDE Jet Brains developed Kotlin, a new programming language for Android applications. Java and Kotlin share many similarities, and switching between the two languages is not difficult. Some of the main differences are as follows:

- Kotlin removes the problem of NullPointerExceptions since all types of variables are non-nullable.
- Kotlin does not have checked exceptions. This means that the developer does not have to declare or catch the exceptions. This approach has both advantages and disadvantages.
- Kotlin allows function extending. This is not supported in Java.
- Java compilation time is approximately 20% faster than Kotlin compilation time.

Java was chosen as the programing language for the presented application. The reason for choosing Java language was mostly the number of publications about Java Android development as well as the amount of available source code and tutorials, which will facilitate the application development process.

5.1.2 Development tools

The primary IDE for Android applications was the Eclipse Android Development Tool (ADT). Eclipse was replaced with Android Studio in 2013, and it is no longer officially supported by Google, although it is still widely used, as it works well with other programming languages. Apart from Android Studio and Eclipse, some companies such as Uber or Spotify use Fabric, a development tool behind Twitter's mobile application. Another option is a GameMaker: Studio, which is popular among game developers. The application provided was developed in the officially supported Android Studio.

5.1.3 Android project structure

This section describes the basic structure of the Android studio project (Figure 5.1). The structure of the project is as follows:

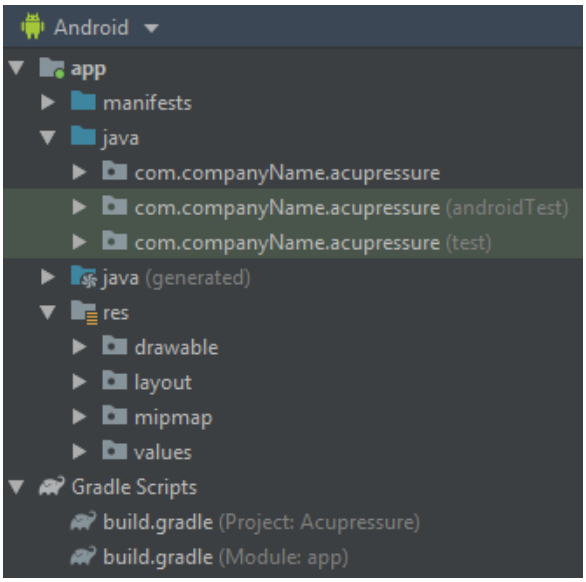


Figure 5.1: Structure of Android Studio project

App

The app folder contains all resources of the application and can be divided into three parts.

The manifests folder contains the AndroidManifest.xml, which holds the basic information about the application. This includes the name of the application, the name of the package, a list of privileges required by the application and a list of the activities and services used in the application, etc.

The Java folder contains Java resource files for the application and the application tests. The usual practice is to divide the application resource

files based on the individual screens of the applications and to add a separate folder for models and general tools such as database manages or REST API clients.

The res folder contains all the remaining resource files, including drawables (images, icons, etc.), layout resource files, or values such as strings and colours used in the application.

Gradle Scripts

Gradle is a build system for Android applications. It is responsible for generating the apk (a runnable android application) from the *.java* and *.xml* files of the project. Gradle scripts are of two types.

The top-level build script is located in the root project directory, and it defines a build configuration for all project modules.

The module-level script contains a definition of the SDK version as well as all the dependencies of the application. The application may contain several module-level scripts, as every local library comes with one module script.

5.2 Basic components of Android application

5.2.1 Activities

An activity is the basic building component for any Android application. Usually, one application screen corresponds to one activity. For example, a gallery app might have one activity for listing all images in the device, another activity to show detail of a selected image, and another activity for modifying images. Only one activity may run at any time. It is possible for one activity to launch another activity via a tool called *intent*. The intent can be understood as a data structure, which holds a description of an action to be performed and may transfer data from one activity to another. For example, when the user clicks on an image in the gallery, the currently running activity puts the ID of the selected image to the intent and asks the intent to start the second activity, which handles displaying the image detail. The second activity is able to extract the ID of the image from the intent to load the image from the database. The topic of the activities is studied in detail in the official documentation [9].

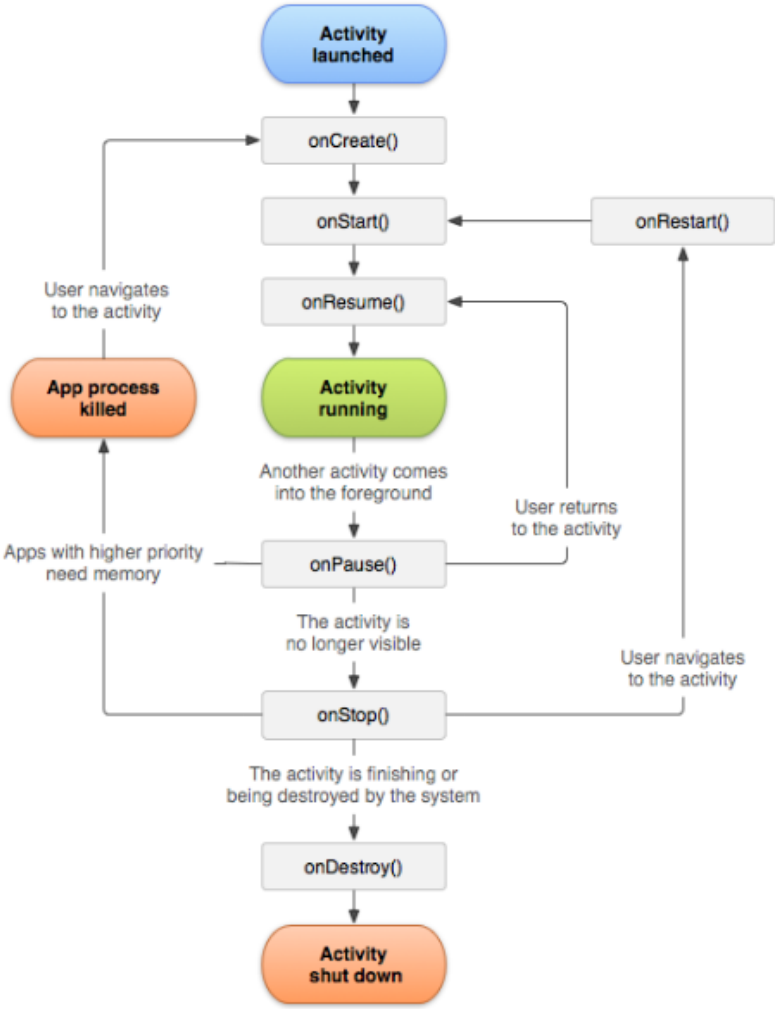


Figure 5.2: Lifecycle of Android activity [9]

The lifecycle of the activity (shown in Figure 5.2) is defined by following callbacks.

- *onCreate()* method, called only once per application lifecycle. It is used to set up the activity, which includes setting the view for the activity, establishing database connection, etc.
- *onStart()* callback may be called multiple times within the application lifecycle. It is called every time the activity becomes visible to the user.
- *onResume()* method, which is called at the moment the activity reaches the top of the activity stack. From this moment, the activity starts to interact with the user.
- *onPause()* method is called when the activity is no longer focusable by the user. It is still visible, but it no longer receives user interaction.

- *onStop()* callback is called when the activity is no longer visible to the user. It is usually used to refresh the UI.

- *onDestroy()* method is the opposite of the onCreate() method. It is the last call before the activity is destroyed, and it is used to free resources used by the activity.

Activities in SelfAcupressure

From now on, the application will sometimes be referred with its product name SelfAcupressure. There follows a list of activities which are used in SelfAcupressure. A detailed description of individual activities can be found in SelfAcupressure application structure section 5.3

- *BaseActivity* (Parent of all other activities, used for activities setup)

- *MainActivity* (Manages the main screen of the app)

- *PointListActivity* (Controls screens displaying any list of points)

- *PointDetailActivity* (Responsible for acupressure point detail screen)

- *ImageDetailActivity* (Manages screen displaying images)

- *PulseDetectorActivity* (Responsible for measuring HR and RR)

- *ShowMeasuredHrActivity* (Displays PulseDetector activity results)

- *AboutAppActivity* (Manages information provision about the app)

- *FeedbackActivity* (Activity which manages app feedback system)

- *SplashActivity* (Activity displaying startup splashart)

5.2.2 Services

A service is an Android tool for handling background operations and interactions with other applications. A service is implemented as a subclass of Service class, and every service has to be declared in the application manifest file. The services might be used, for example, to handle background data synchronization with the server, or to play music in the background, while the user interacts with another application.

SelfAcupressure does not require any custom services. But it uses build-in system-level services, such as *Connectivity service*, which is used to handle network connections.

5.2.3 Broadcast receivers

Broadcast receivers enable notification of the application about events coming from outside of a regular application flow. For example, a broadcast receiver may be used to notify the application about an incoming SMS message. The advantage of broadcast receivers is that the system can even notify applications that are currently not running.

5.2.4 Content providers

Content providers are used for sharing data among multiple applications. This includes, for example, contacts stored in the device, which may be used by multiple applications, and thus they have to be stored in the content provider. As most applications do not have to share the data with other applications, they usually use a simple SQL database to store their data.

5.2.5 Fragments

A fragment is a construct used to divide activities into smaller units. The fragment, in the same way as the activity, has its own life cycle, and it is able to receive its own input events. One activity might be composed of multiple fragments, each responsible for one part of the activity functionality. It is also common to use one fragment in multiple activities, which increases the clarity of the application. There follows a list of activities that are used in SelfAcupressure.

- *Favourite fragment* (fragment which shows a list of the users favourite points)
- *Search fragment* (offers an affliction-based search tool for acupressure points)
- *Body fragment* (manages search tool based on acupressure point location)
- *Advanced fragement* (contains application settings and the secondary application functionality such as feedback entry point or measurement entry point)
- *Tutorial fragment* (contains the vital signs measurements tutorial)
- *Measurement fragment* (fragment where the vital sings estimation is performed)
- *Results fragment* (fragment displaying the results of performed vital sings measurements)

5.3 SelfAcupressure application structure

This section describes the structure of the implemented application in detail. The SelfAcupressure project structure reflects the standard Android application structure defined at the beginning of this chapter (5.1.3). As the basics of the structure are well described above, this section concentrates on the content of the *app/src/main/java* folder (5.3), which contains the functional source code.

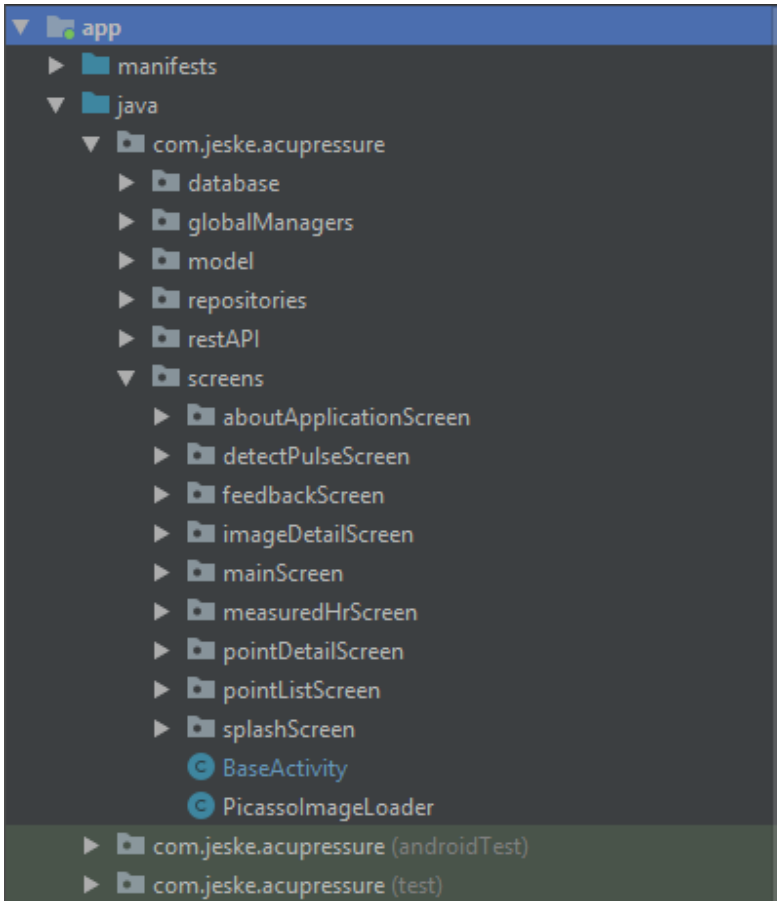


Figure 5.3: Structure of the SelfAcupressure project

The application is at the top level divided into the following packages (as shown in Figure 5.3):

Database package

Database package encapsulates the code necessary to implement MMVM room database (described in Section 4.3.4), which includes:

- *ApplicationDatabase.java* - Class that extend+.s RoomDatabase. It defines which entities the database uses, what the database version is, etc.

- *ApplicationDao.java* - Interface, which contains a definition of all database queries.
- *AplicationDatabaseModule.java* - Class, which handles the proper instantiation of the database, using the Singleton pattern[1].
- *Convertors package* - A package which contains all database convertors[2].

GlobalManages package

A package that comprises classes that provide globally accessible resources. For example, *IntentKeysManager.java*, which holds string resources that identify data passed between activities via intent.

Model package

The Model package contains definitions of all data models used in the application. The most important models are:

- *Point.java*, which defines data models for acupressure points,
- *Problem.java*, which contains models for individial problems solvable by acupressure.
- *Measurement.java*, which defines data objects for HR measurements.

Repositories package

A package which encapsulates MMVM repository classes: *Point repository*, *Problem repository* and *HeartRate repository*. Each managing CRUD[3] operations for corresponding data objects.

RestApi package

Like the Database package, the RestApi package encapsulates the code required by the Retrofit library to create a rest API client for communication with the remote server, which includes:

- *APIClient.java* - Class that defines the communication point (URL of the remote server, etc.).
- *APIInterface.java* - Interface, which contains a definition of rest API queries.
- *DataProcessor.java* - A custom class that contains methods for processing data from incoming rest communication.

[1] software design pattern restricting the instantiation of a class to one instance

[2] converts complex Java structures (Lists, Maps, etc.), to simple data structures, used by the room database

[3] CRUD (Create Read Update Delete), basic operations applied on data objects

Screens package

A package that serves as a container for source code of individual application screens. Each screen is placed in a separate package as follows:

- *SplashScreen* - Simple screen, that displays the SelfAcupressure logo, at the application startup.
- *MainScreen* - The core of the application. This screen is divided into four fragments, each managing one functionality. The user is provided with a bottom navigation bar to switch between fragments.
  - *Favourite fragment* - contains a list of the users favourite points
  - *Search fragment* - offers an affliction-based search tool
  - *Body fragment* - manages search tool based on acupressure point location
  - *Advanced fragement* - encapsulates additional application functionality, such as application settings, feedback entry point or measurement entry point

  For clarity, each fragment is placed in its own package, which holds its Java class as well as ViewModel and other auxiliary classes provided.
- *PointListScreen* - This screen is accessible from the Search and Body fragment, and displays a list of points that were filtered by one of the search tools.
- *PointDetailScreen* - The task of this screen is to display detailed information about any filtered acupressure point.
- *ImageDetailScreen* - Contains a tool to display zoomable detail of acupressure point images.
- *DetectPulseScreen* - Similarly to MainScreen, the DetectPulseScreen is divided to three fragments each responsible for one functionality.
  - *Tutorial fragment* - displays a tutorial describing the measurements procedure
  - *Measurement fragment* - fragment where the measurement itself is performed
  - *Results fragment* - contains a list of performed measurements, and the entry point to MeasuredHrScreen
- *MeasuredHrScreen* - Activity that displays the results of the currently performed measurement or results of previous measurements
- *AboutApplicationScreen* - Simple activity that describes the application background
- *FeedbackScreen* - Serves as an entry point to send feedback via Android build in communication means, such as E-mail or Gmail

5.4 Used tools and libraries

This section describes the tools and libraries used in the SelfAcupressure application. As mentioned earlier, Android applications are built by the Gradle build system. This system makes it easy to include external binaries and other library modules in the application. For most libraries, the only necessary step to use a given library is to declare a dependency on that library in the build.gradle file. SelfAcupressure uses the following libraries: Room, Retrofit, Dagger, OpenCV, OkHttp, Gson, Picasso, PhotoView, and MaterialShowcaseView.

5.4.1 Room

As the application implements the MMVM architecture, Room library[10] is a necessary component. As stated in the official documentation: "The Room persistence library provides an abstraction layer over SQLite to allow robust database access while harnessing the full power of SQLite."

5.4.2 Retrofit

Based on the official Android documentation, the Retrofit library[11] is the currently recommended tool for REST API connections. Thanks to the Retrofit toool it is possible to declare HTTP methods inside the application in the Java language. An example of this method is as follows:

```
@GET("serializer")
Call<MultipleResource> doGetMultipleResource(@Query("lang") String
    lang);
```

5.4.3 OkHttp

OkHttp[12] is an open-source library that is used by the Retrofit tool to establish an HTTP connection. OkHttp is able to recover from common connection problems and preserve the connection in case of a network outage.

5.4.4 Dagger

The Dagger tool[13] is a static framework for compile-time dependency injection [4]. In MMVM architecture, the Dagger tool is used to provide Repository instancies throughout the application. The constructor of every repository is annotated with @Inject annotation. When a ViewModel requires dependency on a given repository, it can obtain its instance with the following call:

[4] dependency injection is a technique where one object supplies the dependencies of another object

```
RepComp repositoryComponent =
    DaggerRepComp.builder().applicationDatabaseModule(new
    ApplicationDatabaseModule(application)).build();
Repository repository = repositoryComponent.getRepository();
```

Using Dagger guarantees that every ViewModel has access to the same repository. Moreover, every implementation modification of that repository is automatically propagated to all ViewModels.

5.4.5 OpenCV

The OpenCV tool[14] is a library with its primary focus on real-time computer vision applications. It is written in C++ with an interface primarily aimed for the same language, but the library also provides bindings to other standard programming languages such as Python, Java, Matlab, or even less common languages as Perl, Haskell or Ruby. The OpenCV library provides tools for a wide range of computer vision topics, such as facial recognition, object identification, motion tracking, and much more. SelfAcupressure uses OpenCV for heart and respiratory rate measurements, which are studied in detail in the next chapter.

5.4.6 Gson

Gson[15] is a widely used library for JSON object serialization. In order to serialize an Android object, it is necessary to annotate its fields with *@SerializedName("nameOfTheField")* annotation, which defines how to map the Android object to JSON data structure. SelfAcupressure uses the GSON library to deserialize incoming data from the remote server.

5.4.7 Picasso

The Picasso library[16] is a powerful tool for downloading and caching images in Android applications. It simplifies the process of downloading images from remote storage greatly. The library requires simply a URL location of the remote image. With the provided URL, the Picasso downloads the image on a background thread, and notifies the image holder, when the image is ready. The library also stores the downloaded images in a local repository so that the following requests can be handled without communication with the remote server.

5.4.8 PhotoView

The PhotoView library[17] is a small tool that provides easily usable implementation of a zooming Android ImageView. The application uses this feature to make every image describing the location of acupressure points, zoomable.

5.4.9 MaterialShowcaseView

The MaterialShowcaseView[18] is a library that provides tools to highlight and showcase specific parts of the application to the user, with a distinctive and attractive overlay. It is a great way to provide an application tutorial or set of hints to highlight less intuitive features or gestures.

Chapter 6

Vital signs estimation

6.1 Introduction

In the past decade, smartphones have moved from simple tools focused on communication via calls and SMS, to complex devices, which can perform a wide range of activities. Nowadays, an average smartphone can be used as navigation, camera, dictionary, credit card, health care assistant, and much more. That healtcare is so prevalent is confirmed by the number of applications on Google Play for fitness assistance, health coaching, vital signs monitoring, etc. The goal of this chapter is to analyze the possibilities for functionality that was yet not addressed by many mobile applications, vital signs monitoring, namely, estimation of heart rate (HR) and respiratory rate (RR).

6.2 Related work

Vital signs monitoring is essential for clinical diagnostics and fitness coaching. Vital signs are usually measured with electrocardiography sensors or photoplethysmography (PPG) sensors. Measurements with these sensors require direct contact with the skin of the person being examined, which may be inconvenient and inappropriate for long time measurements. Therefore, a number of scientific groups have conducted studies to come up with algorithms for remote vital signs monitoring. A large portion of these studies examines the possibility of vital signs estimation using camera footage. Algorithms which estimate vital signs from video footage are ideal for mobile devices, as they only require a camera and no other sensors to perform the measurements.

The first papers dedicated to vital signs monitoring were aimed primarily at HR estimation from facial videos. This method is known as remote photoplethysmography (rPPG). The principle of rPPG is to observe colour changes in the subject's skin. The skin colour reflects changes in the blood volume in capillaries, which is consistent with the pulse signal. The process of rPPG is well described in [19] and [20]. The basics of rPPG methodology and a general overview of all papers dedicated to rPPG is presented by Rouast,

Philipp et al.[21]. rPPG can also be used to estimate the respiratory rate, as RR in addition influences the volume of blood in capillaries. However, RR detection is more complicated and less precise, as will be shown in the following pages.

Another approach to estimate RR from video footage of a person is body motion detection. For example in papers [22, 23, 24] the chest motion of tested subjects is tracked to measure their RR. Another interesting paper, published by Rubinstein at al. [25], shows how face motions can be used for vital signs estimation. Their algorithm observes face expansion and shrinkage, which is again caused by the changing volume of blood in capillaries. These changes, when amplified, can be used to estimate the HR signal.

6.3 Vital signs estimation on Android

The previous section introduced two approaches on how to use camera recording to estimate the vital signs of a person. The algorithms for chest motion detection usually require a static shot taken from a greater distance, so the whole chest region is included in the field of view. That is hard to achieve while holding the camera in the hand and, therefore, not very suitable for mobile applications. However, making a facial recording is simple as most smartphones today are equipped with a front camera. Based on these assumptions, the chest motion approach was rejected, and the photoplethysmography approach was subjected to a more detailed analysis.

The system for estimating the HR/RR from the video of the face can be described with the following diagram 6.1.

Figure 6.1: Video processing scheme

The process starts with examining each frame individually. Initially, a face is detected and extracted from each frame. Subsquently, the facial image is subjected to decomposition into its RGB channels. As discussed above, each channel reflects changes in the blood volume in facial capillaries, which are influenced by both RR and HR signals. Usually, the red channel is used for the following spectral analysis as it best reflects blood volume changes. For each frame, the system calculates an average value of the red colour across all the pixels. From these values, a signal representing the color changes is composed. The last step is to identify the frequency of the RR/HR in the red channel signal, which is a task of simple signal processing. The signal is converted to a frequency domain by means of a Fourier transform. The HR can be located as the highest peak of the frequency domain in the interval from 0.8 to 2.5 Hz, which are the boundaries of an average healthy person's heart rate. The RR should be similarly detectable as the highest peak in the interval from 0.15 to 0.45 Hz.

Although many studies were dedicated to facial recording analysis, this approach has several drawbacks. Firstly the results of the measurement may be affected by ambient lighting, as some light sources flash while running. Flashing ambient light may distort the recording, and the estimated HR frequency may actually be the frequency of the flashing light. The second problem is computational complexity, which comes with the face extraction. In order to offer real-time measurements, computational complexity should be as low as possible. The third drawback can be the measurement process itself because the user has to face the phone directly throughout the measurement, which is not very user friendly, and may discourage some users from using the application.

The solution to the problems outlined is fairly simple. The measurement process can be modified by replacing facial footage for footage of a finger, as shown in the following figure 6.2. The finger is placed over the rear camera and its flashlight, and video footage is taken with flashlight turned on. The finger can be used instead of a face, as it also contains the heart and respiratory signals hidden in the amount of blood flowing through the finger. In this case, no ambient light influences the measurement because the finger fully covers the camera lens. The complexity of the evaluation process is decreased, as there is no facial extraction performed. In addition, from the users point of view, this approach is more comfortable, because the user does not have to look at the camera throughout the measurement. The applicability of this approach confirms several applications recently published on Google Play, which already use the rear camera with the flash to estimate users HR [26, 27]. Based on this analysis, this estimation method was selected as the best approach to vital signs estimation.

Figure 6.2: Recording finger footage [28]

6.4 Vital signs estimator implementation and evaluation

The video analyzer was implemented using the OpenCV library. The reason for using the OpenCV library for video footage analysis is its wide set of tools for image processing. OpenCV enables processing the video footage frame by frame in real-time. Each frame is published in a standard OpenCV Mat format, which is a multidimensional matrix representation of the frame optimized for fast calculations. Thanks to this matrix representation, the decomposition of frames to its RGB channels is fast and straightforward. The signal processing methods were implemented using the OpenCV library and standard Apache math library.

There follows an evaluation of the presented estimator accuracy. Accuracy evaluation is divided into two parts. The first part examines the accuracy of the HR estimation compared to an OMRON M6 digital oscilometer [29] and another android application for HR measurement, the Heart Rate Plus [26]. The second part contains a discussion about RR estimation and an evaluation of its accuracy.

6.4.1 Heart rate estimation

A series of measurements were performed to evaluate the precision of the HR estimation. The measurements were carried out at different times of the day in various conditions (after exercise, while resting, after drinking coffee, etc.). HR was measured simultaneously with the SelfAcupressure estimator installed on a Samsung Galaxy A5 and with an OMRON M6 oscilometer [29]. The same procedure was subsequently used to evaluate the accuracy of the Heart Rate Plus [26]. The results are presented in the following tables 6.1, 6.2 .

No.	Omron M6[b/s]	SelfAcupressure[b/s]	Deviation[b/s]
1	55	55	0
2	53	53	0
3	52	55	3
4	54	56	2
5	73	69	4
6	53	53	0
7	88	93	0
8	65	67	2
9	71	71	0
10	71	74	0
11	78	74	4
12	72	72	0
13	72	70	2
14	55	53	2
15	59	60	1

Table 6.1: SelfAcupressure HR estimator compared to Omron M6 oscilometer

No.	Omron M6[b/s]	Heart rate plus[b/s]	Deviation[b/s]
1	71	72	1
2	68	71	3
3	75	78	3
4	72	76	4
5	72	72	0
6	67	67	0
7	68	66	2
8	58	59	1
9	59	58	1
10	85	81	4
11	55	57	2
12	63	61	2
13	49	50	1
14	70	70	0
15	50	52	1

Table 6.2: Heart rate plus estimator compared to oscilometer Omron M6

Table 6.1 shows that the SelfAcupressure estimator returns consistent results, which correspond to HR values from Omron M6 with an average deviation of 2 b/min, which is approximately the same accuracy as the examined Heart rate plus as shown in the table 6.2.

6.4.2 Respiratory rate estimation accuracy

The previous section shows that the finger color variability is a sufficient indicator to measure heart rate, as the HR is clearly detectable in the frequency spectrum. On the other hand, the respiratory rate turned out to have only a small effect on the amount of blood flowing through the finger, and its estimation is, therefore, more complicated. The following graph 6.3 shows the frequency spectrum from one of the experiments. Whereas the peak, which corresponds to the heart rate frequency, is clearly visible (purple circle), the red peak, which indicates the respiratory rate is relatively small compared to other signal components. In this case, the evaluation system would incorrectly estimate the RR, as it would choose the peak next to the actual RR as the dominant frequency component in the given interval (0.15, 0.45 Hz). For this reason, the RR estimation displays a higher inaccuracy than HR estimation, because in some cases, the effect of breathing is drowned out by noise.

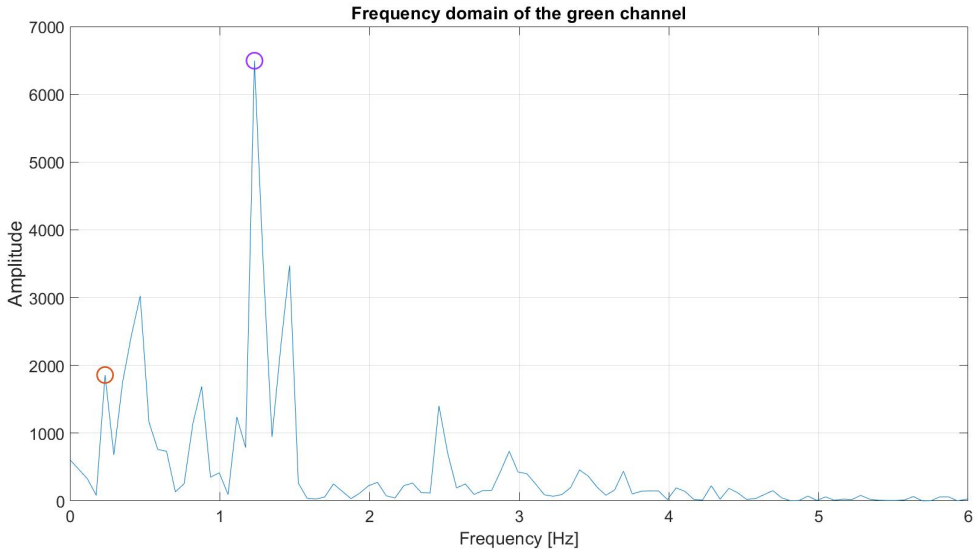


Figure 6.3: Frequency spectrum of the red chanel demonstrating the difference of HR and RR discoverability

To calculate the accuracy of the RR estimation, a series of experiments were performed. In these experiments, the tested subject was asked to breathe at regular intervals while his RR was measured using the SelfAcupressure estimator. The following table 6.3 shows the results of the experiments performed. The average deviation is approximately 3 br/min, which is acceptable compared to the state-of-the-art accuracy, which, based on methodology, ranges from 1.2 - 2 br/min.

No.	Breathing frequency[br/s]	Estimated RR[br/s]	Deviation[br/s]
1	12	13	1
2	12	13	1
3	12	11	1
4	12	15	3
5	15	14	1
6	15	13	2
7	15	12	3
8	15	13	2
9	20	17	3
10	20	15	5
11	20	17	3
12	20	15	5
13	10	13	3
14	10	13	3
15	10	18	8

Table 6.3: Results of experiments to evaluate RR estimation accuracy

6.5 Implementation example

Here follows an example of estimation implementation (Figure 6.4) to demonstrate the implementation approach and style of writing the code. The code provided contains the implementation of the *calculateVitalSings()* method used to estimate HR and RR from a list of video frames.

```
224 @   private VitalSigns calculateVitalSings() {
225
226         Scalar[] framesToEvaluate = getFramesForLastXSeconds();
227
228         double[] red = getColorFromScalar(framesToEvaluate, colorIndex: 0);
229
230         double[] redWithoutStaticComponent = substractAvg(red);
231         Spectrum powerSpectrum = getSignalPowerSpectrum(redWithoutStaticComponent, FRAME_RATE);
232
233
234         double hb_frequency = getDominantFrequencyInRange(powerSpectrum, FRAME_RATE,
235                 redWithoutStaticComponent.length, HR_LOWER_BOUND, HR_UPPER_BOUND);
236         int hr = (int) Math.round(hb_frequency * 60);
237
238         double rr_fr = getDominantFrequencyInRange(powerSpectrum, FRAME_RATE,
239                 redWithoutStaticComponent.length, RR_LOWER_BOUND, RR_UPPER_BOUND);
240         int rr = (int) Math.round(rr_fr * 60);
241
242         return new VitalSigns(hr, rr);
243     }
```

Figure 6.4: Implementation example demonstrated on the vital sings evaluation method

At first, the method requests the frames to process (line 226) in the form of a Scalar list (standard OpenCV object for image representation). Then the red channel is extracted from the list (line 228). Follows the removal of the static component from the red channel (standard procedure before

applying the Fourier transformation) and the power spectrum extraction from the remaining signal (lines 230 and 231). In the last step, HR and RR are extracted from the power spectrum and returned from the method in the form of VitalSings object.

6.6 Summary

This chapter presented the problem of heart rate and respiratory rate estimation on android devices. The experiments in this chapter showed that the implemented HR estimation achieves satisfactory results and can compete with existing android applications offering this functionality. It is to be decided if the SelfAcupressure application should also provide RR estimation functionality, which does not achieve such quality results.

Chapter 7

User testing

This chapter contains a summary of user testing, which was performed to assess user acceptance of the application before the application is placed on the market. User testing was performed on a sample of five potential users. This sample was chosen to contain a full range of potential users, both men and women aged from 20 to 60 years, as shown in the following table.

Gender	Age	occupation
Female	20	undergraduate student
Male	59	manager in IT company
Female	59	high school teacher
Male	21	undergraduate student
Female	49	household management

Table 7.1: User profiles

At the beginning of the test, each user was briefly acquainted with the focus of the application. Then the users were presented with a set of tasks to perform using the application. Examples of individual tasks are as follows:

- You have recently experienced digestive problems. Open the application and try to find the acupressure points that could help you with this problem.
- Choose one of the points for dealing with digestive problems and find out as accurately as possible where this point is located and how it is recommended to stimulate this point.
- Measure your current heart rate with the application. When the measurement is complete, return to the screen where you performed the measurement.

The entire set of tasks can be found in the appendix E. Although each of the tests was performed in the presence of the application developer, the testing process was not interfered with in order to observe behaviours and to detect any deficiencies in the application. At the end of the testing, each user was prompted to comment on the application and suggest any application modifications that would improve the user experience.

7.1 User test results

7.1.1 Observations summary

As expected, young respondents showed themselves to be more skilled and intuitive while operating the application. Both of them passed the tests without any problems. The only problematic step was to perform the heart rate measurement, as they started the measurement without reading the tutorial and then had difficulty operating the tool. This finding led to the addition of a tutorial, which is displayed the first time the user enters the screen for measurements and which prompts the user to start with the tutorial. The older respondents encountered two other problems. At first, they had difficulty navigating the application as they were not used to the bottom navigation bar. This problem was resolved with an onboarding tutorial, which shows the user how to use the bottom navigation bar, and how the app is organized. The second problem turned out to be the advanced tab on the main screen. Here the users have trouble distinguishing which UI elements are interactable and which are not. This observation led to a rework of the screen with a more evident division between active and passive UI elements.

7.1.2 User feedback

All respondents agreed that they would welcome more instructional texts to make it easier for them to work with the application. Except for minor difficulties with orientation in the application, all the users found the app interesting, and they could imagine using the application once available.

Chapter 8

Conclusion

The presented thesis dealt with designing and implementing an application for self-acupressure. The application aims to teach the user how to practice acupressure and to serve as a database of acupressure points. Previous chapters describe in detail the process of developing the Android application, from assembling the initial concept in Balsamiq Mockups and Adobe XD, though the application implementation in Android Studio, to user testing of the final product. A demonstration of the resulting application is given in appendix D.

A part of the development process was the implementation of the application server, which serves as the remote database for the application. The server was implemented in Python using the Django tool. The application uses the server once a day to synchronize data and when necessary to obtain images of acupressure points and exercises.

Chapter 6 was devoted to vital signs estimation to examine the possibilities of using todays smartphones in the field of heart rate and respiratory rate estimation. The reason for this analysis was the idea of incorporating these features into the application to offer the user an additional functionality that would elevate the application above other applications. The analyses performed showed that it is possible to implement both functionalities using the camera of the phone.

The tests measuring the accuracy of the implemented solution showed that the application is able to measure HR with an accuracy of two beats per minute and RR with accuracy of three breaths per minute, which is sufficient to include the estimators to the application.

8.0.1 Further application development

The application is now ready to be placed on the market. Nevertheless, the current implementation still leaves some space for further improvements. There follows a list of some possible application enhancements which could be implemented in the future:

- The application could contain a set of screens dedicated to acupressure in general. On these screens, the user could learn about the origins of the acupressure or about the current acupressure trends.
- Another possible improvement could be instructional videos. These videos could be about the acupressure in general, or they could complement the existing image gallery in describing the position and stimulation of acupressure points.
- With the recent progress of virtual reality, the application may also contain 3D models of individual body parts with highlighted acupressure points for easier point localization.

Appendix A

List of abbreviations

Symbol	Meaning
HR	Heart rate
RR	Respiratory rate
rPPG	remote photoplethysmography
UI	user interface

Appendix B

Bibliography

[1] Operating System Market Share. https://gs.statcounter.com/os-market-share/mobile/. Accessed: 2020-02-10.

[2] The Google Play store. https://play.google.com/store. Accessed: 2020-02-16.

[3] Acupressure Point for 300+ Diseases. https://play.google.com/store/apps/details?id=ethanapp.acupointsceng. Accessed: 2020-02-18.

[4] Basics of Acupressure Massage. https://play.google.com/store/apps/details?id=com.acupressure.points.acupressurebasics. Accessed: 2020-02-10.

[5] Acupuncture Index. https://play.google.com/store/apps/details?id=com.acupressure.points.acupressurebasics. Accessed: 2020-02-10.

[6] Balsamiq mockups. https://balsamiq.com. Accessed: 2020-01-01.

[7] Adobe XD. https://www.adobe.com/products/xd.html. Accessed: 2020-01-01.

[8] Guide to app architecture. https://developer.android.com/jetpack/docs/guide. Accessed: 2020-02-11.

[9] Android activity documentation. https://developer.android.com/reference/android/app/Activity/. Accessed: 2020-02-16.

[10] Room Persistence Library. https://developer.android.com/jetpack/androidx/releases/room. Accessed: 2020-04-14.

[11] Retrofit Library. https://square.github.io/retrofit. Accessed: 2020-04-14.

[12] OkHttp Library. https://square.github.io/okhttp. Accessed: 2020-04-14.

[13] Dagger tool. https://dagger.dev. Accessed: 2020-04-14.

[14] OpenCV library. https://opencv.org. Accessed: 2020-04-14.

[15] Gson library. https://github.com/google/gson. Accessed: 2020-04-14.

[16] Picasso image library. https://square.github.io/picasso. Accessed: 2020-04-14.

[17] PhotoView library. https://github.com/chrisbanes/PhotoView. Accessed: 2020-04-14.

[18] MaterialShowcaseView library. https://github.com/deano2390/MaterialShowcaseView. Accessed: 2020-04-14.

[19] M. Poh, D. J. McDuff, and R. W. Picard. Advancements in noncontact, multiparameter physiological measurements using a webcam. *IEEE Transactions on Biomedical Engineering*, 58(1):7–11, 2011.

[20] W. Wang, A. C. den Brinker, S. Stuijk, and G. de Haan. Algorithmic principles of remote ppg. *IEEE Transactions on Biomedical Engineering*, 64(7):1479–1491, 2017.

[21] Philipp Rouast, Marc Adam, Raymond Chiong, David Cornforth, and Ewa Lux. Remote heart rate measurement using low-cost rgb face video: A technical literature review. *Frontiers of Computer Science (electronic)*, 12:858–872, 09 2018.

[22] K. S. Tan, R. Saatchi, H. Elphick, and D. Burke. Real-time vision based respiration monitoring system. In *2010 7th International Symposium on Communication Systems, Networks Digital Signal Processing (CSNDSP 2010)*, pages 770–774, 2010.

[23] Rik Janssen, W. Wang, Andreia Moço, and Gerard Haan. Video-based respiration monitoring with automatic region of interest detection. *Physiological measurement*, 37:100–114, 12 2015.

[24] K. Lin, D. Chen, and W. Tsai. Image-based motion-tolerant remote respiratory rate evaluation. *IEEE Sensors Journal*, 16(9):3263–3271, 2016.

[25] Hao-Yu Wu, Michael Rubinstein, Eugene Shih, John Guttag, Frédo Durand, and William Freeman. Eulerian video magnification for revealing subtle changes in the world. *ACM Transactions on Graphics - TOG*, 31, 07 2012.

[26] Heart Rate Plus application. https://play.google.com/store/apps/details?id=com.dungelin.heartrate. Accessed: 2020-04-24.

[27] Heart Rate Monitor application. https://play.google.com/store/apps/details?id=com.repsi.heartrate. Accessed: 2020-04-24.

[28] How to measure heart rate. https://help.runtastic.com/hc/en-us/articles/203502962-How-to-measure-my-heart-rate. Accessed: 2020-04-24.

[29] Blood pressure monitor OMRON M6. https://www.medisave.co.uk/omron-m6-comfort-blood-pressure-monitor.html. Accessed: 2020-04-24.

Appendix C

Adobe XD Wireframes

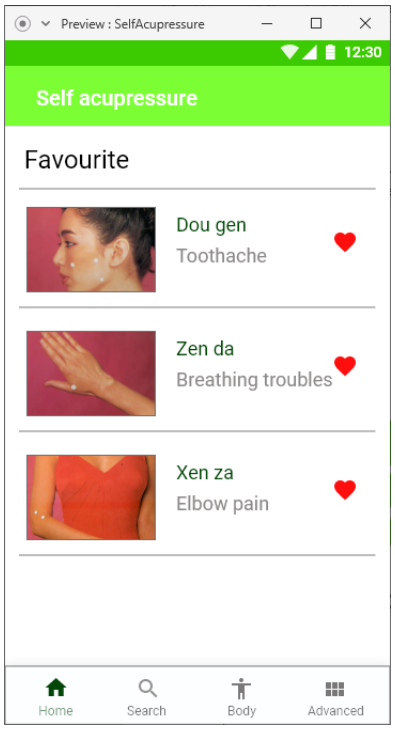


Figure C.1: Favourite screen

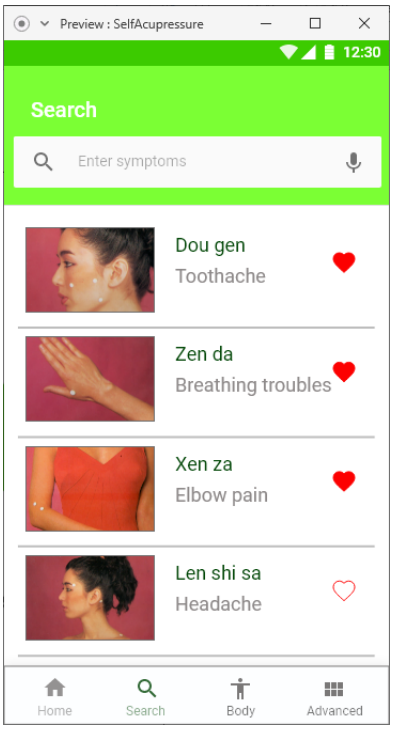


Figure C.2: Search screen

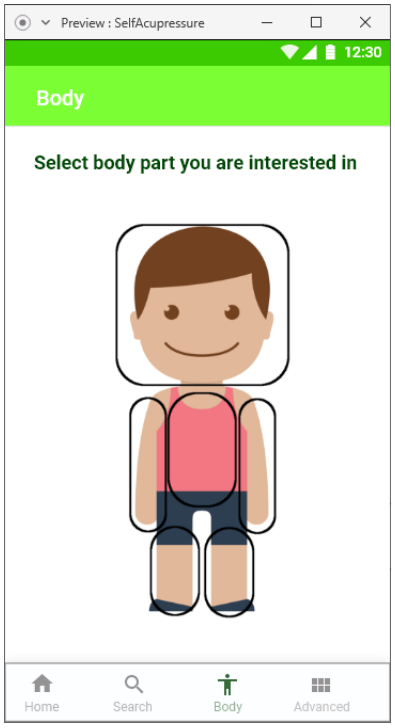


Figure C.3: Body screen

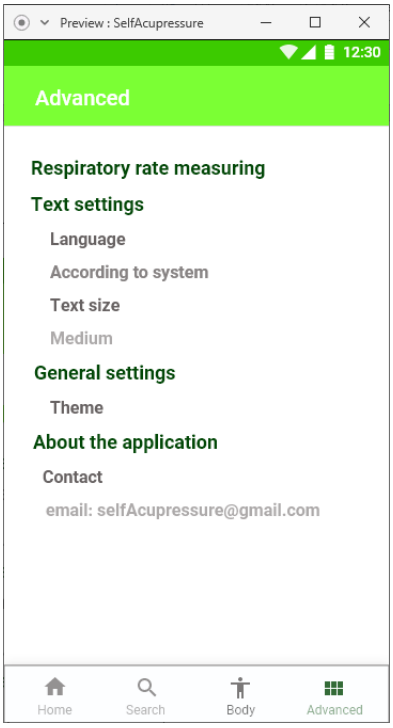


Figure C.4: Advanced screen

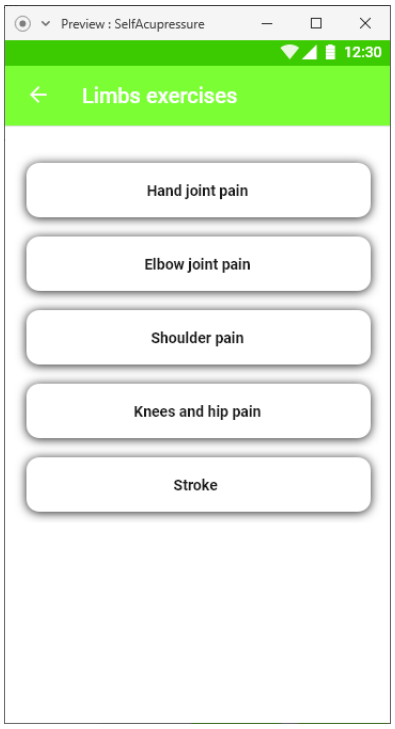


Figure C.5: Point list screen

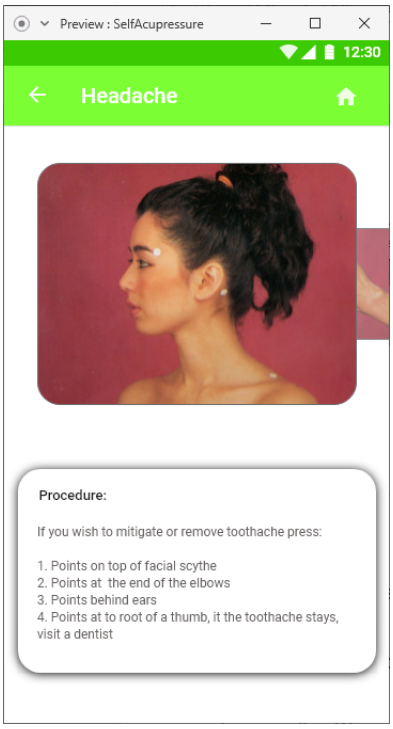


Figure C.6: Point detail screen

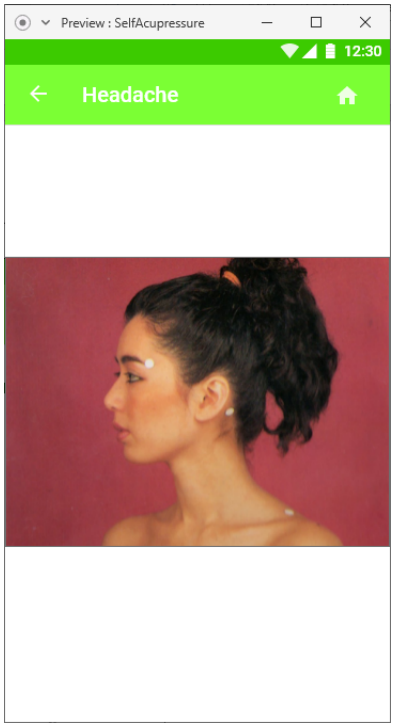


Figure C.7: Image detail screen

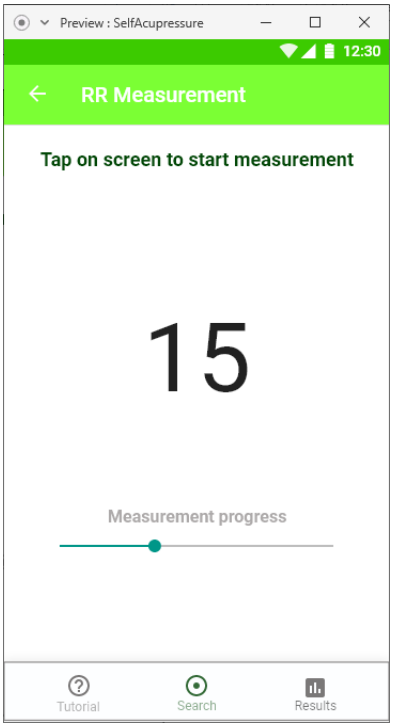


Figure C.8: RR measurement screen

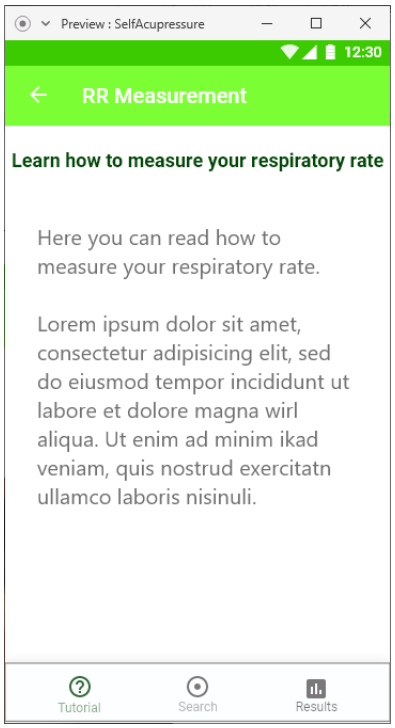


Figure C.9: RR tutorial screen

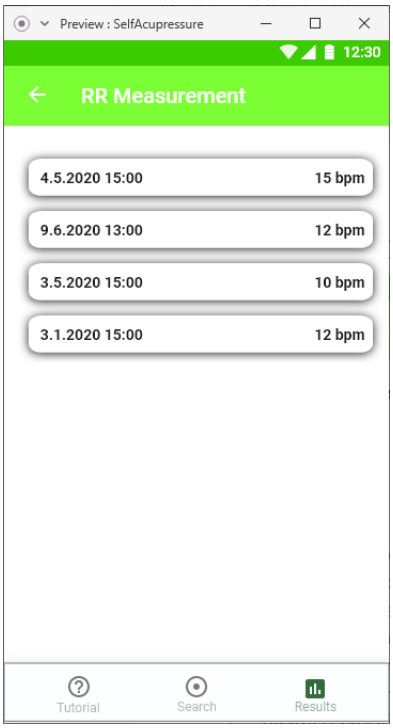


Figure C.10: RR rersults screen

Appendix D

Application demonstration

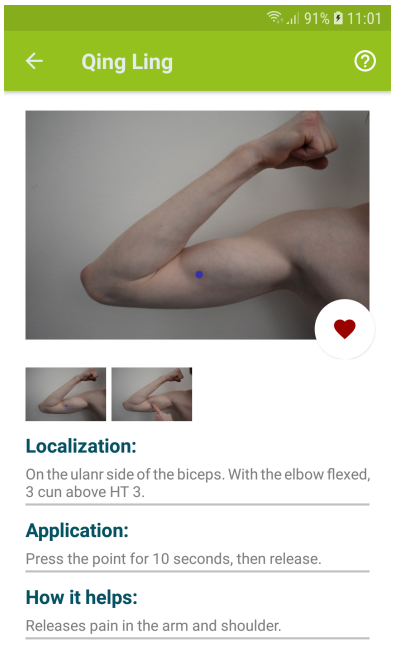


Figure D.1: Point detail screen

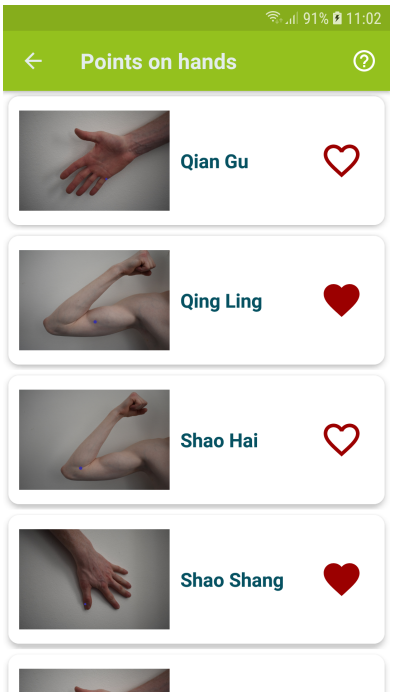


Figure D.2: Point list screen

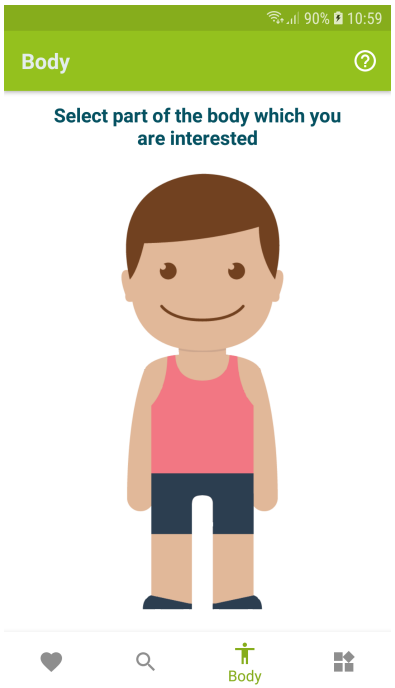


Figure D.3: Body screen

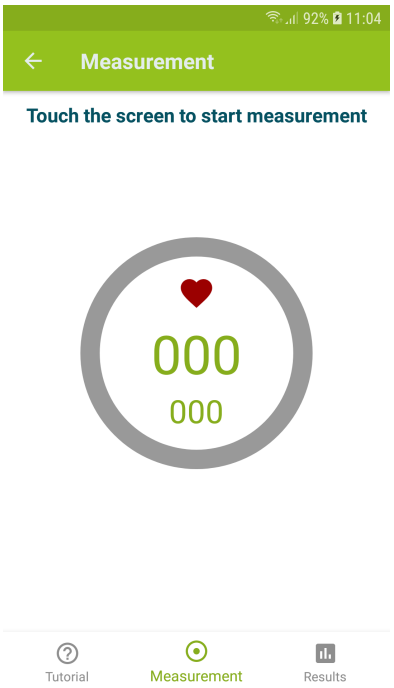


Figure D.4: Measurements screen

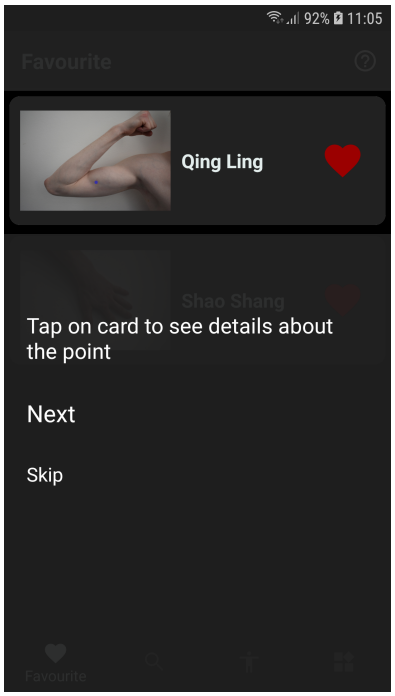


Figure D.5: Favourite screen

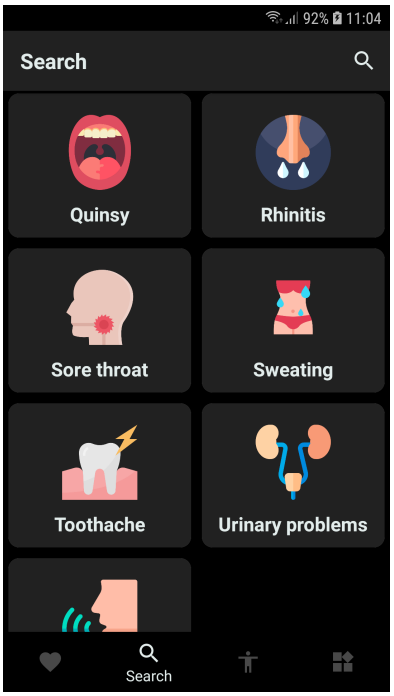


Figure D.6: Search screen

Appendix E

User tests

1. You have recently experienced digestive problems. Open the application and try to find the acupressure points that could help you with this problem.
2. Choose one of the points for dealing with digestive problems and find out as accurately as possible where this point is located and how it is recommended to stimulate this point.
3. You would like to know which points are located on your hands. Try to display a list of all these points, and subsquently, choose one of those points and open the point detail.
4. You are not sure about the exact position of the point. Try to view a detail of the point image.
5. Look at details of the other images.
6. Return back to the point detail. You are not sure what the heart button is for. Display the prompter, which explains the purpose of the button.
7. You would like to send an improvement suggestion to the author of the application. Try to send a message to the author, in which you suggest a colour change of the application theme to blue.
8. You would like to use the application at night. Try to switch the application to night mode.
9. Measure your current heart rate with the application. When the measurement is complete, return to the screen where you performed the measurement.
10. You would like to know the results of the last performed measurement. Try to locate it in the application.