

I. Personal and study details

Student's name: **Klouček Štěpán** Personal ID number: **457180**
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:

iOS application for acupressure

Master's thesis title in Czech:

iOS aplikace pro akupresuru

Guidelines:

The goal of the diploma thesis is to implement an iOS application, which helps the user to solve their common medical problem through acupressure.

- 1) Get basic knowledge of acupressure and analyze the existing applications for iPhone.
- 2) Create a design of the application, which will be coherent with the iOS guidelines.
- 3) Implement application by created design.
- 4) Try to use and implement augmented reality to simplify localization of acupressure points
- 5) Verify the functionality of the application with 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

Diploma thesis



Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Computer Science

iOS Application for Acupressure

Bc. Štěpán Klouček *et al.*

Supervisor: Ing. Václav Burda
May 2020

Acknowledgements

I would like to thank everyone who has supported me during my studies and helped me with this thesis. Namely, I would like to thank my girlfriend Kateřina for being supportive, Daniel Novák and Václav Burda for their innovative ideas and Jindřich Prokop for his help with the server. Finally, I would like to express gratitude to my family for their continuous emotional and financial support.

Declaration

I declare that this work is all my own work and I have cited all sources I have used in the bibliography.

Prague, May 21, 2020

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

V Praze, 21. května 2020

Abstract

The goal of this thesis is to design and build a mobile application for iOS, which aids the user by addressing common medical problems through augmented reality acupressure. Included in this thesis is an analysis of already existing solutions, as well as a prototype of the user interface which accommodates the iOS guidelines. The application was built using SWIFT programming language and implements an augmented reality feature to simplify the user's experience with acupressure. Through augmented reality, the user is able to easily view and localize acupressure points and self-apply this treatment. The completed application has been tested on six users.

Keywords: acupressure, iOS, augmented reality, VIPER

Supervisor: Ing. Václav Burda

Abstrakt

Cílem této práce bylo vytvořit mobilní aplikaci pro systém iOS, která uživateli pomůže s jeho zdravotními problémy pomocí akupresury. Součástí práce je analýza existujících aplikací. Návrh a design uživatelského rozhraní tak aby odpovídal iOS standartům. Podle vytvořeného návrhu byla implementována aplikace v jazyce SWIFT. Pro zjednodušení práce s akupresurními body byla do aplikace implementována rozšířená realita. Ta dokáže být cenou pomocí především při lokalizování akupresurních bodů. Celá aplikace byla nakonec otestována šesti uživateli.

Klíčová slova: akupresura, iOS, rozšířená realita, VIPER

Překlad názvu: iOS aplikace pro akupresuru

Contents

1 Introduction	1	6.3.3 Screen modules	35
1.1 Acupressure	1	6.4 Dark mode	35
1.2 iOS application	2	6.5 Resources	37
1.3 Structure	3	6.5.1 Localisation	37
2 Existing solutions	5	6.5.2 Images	37
2.1 Existing apps analyze	8	7 Augmented reality	39
2.1.1 Acupressure: Heal yourself . .	8	7.1 Basics	39
2.1.2 Lite apps	9	7.2 AR and acupressure	40
2.1.3 Acupressure: Self Healing massage therapy	10	7.3 Frameworks	41
2.2 Conclusion	10	7.4 Implementation	42
3 Application functionality	11	7.4.1 Plane detection	42
3.1 Functional requirements	11	7.4.2 Image detection	42
3.2 Non-functional requirements . .	12	7.4.3 User interaction	42
3.3 Use cases	13	7.4.4 Showing detail	43
4 Development for iOS	15	8 User testing	45
4.1 Language choice	15	8.1 First iteration	46
4.2 Swift	15	8.2 Second iteration	47
4.2.1 Design patterns	16	8.3 Future work and improvements	48
4.2.2 Arc	16	9 Conclusion	49
4.2.3 Libraries	18	A Bibliography	51
4.2.4 Distribution	18	B Design	53
4.2.5 Architecture Patterns	19	C Self acupressure screenshots	57
5 Design and UX	23		
5.1 Navigation and flow	24		
5.2 Design	25		
5.2.1 Onboarding	25		
5.2.2 Main screen	26		
5.2.3 Search screen	26		
5.2.4 Settings screen	26		
5.2.5 Detail screen	27		
5.3 Design and implementation differences	27		
5.3.1 Detail screen changes	28		
5.3.2 Settings screen changes	28		
5.3.3 Search screen	28		
6 Implementation	31		
6.1 Cocoa pod usage	31		
6.1.1 R.swift	32		
6.1.2 Stevia Layout	32		
6.1.3 RxSwift	33		
6.2 BR-VIPER	33		
6.3 Architecture overview	35		
6.3.1 Dependencies	35		
6.3.2 Repositories	35		

Figures

1.1 One of acupressure points used in application.	2
2.1 Applications using same template with different name to increase its reach	7
2.2 Acupressure: Heal yourself application look	8
2.3 "Lite" acupressure application look	9
2.4 Acupressure: Self Healing massage therapy application screen examples	10
4.1 Simple retention cycle	16
4.2 Retention cycle in VIPER	17
4.3 Rising memory indicating possible memory problem	17
4.4 MVC architecture pattern [2]...	20
4.5 MVVM architecture pattern [18]	20
4.6 VIPER architecture pattern [8] .	21
5.1 Navigation in self acupressure application.....	24
5.2 Navigation flow of Self-acupressure application.....	25
5.3 Self acupressure onboarding design	26
5.4 Landing page of application ...	27
5.5 Landing page of the application	28
5.6 Settings screen design and implementation comparison	29
5.7 Search screen design and implementation comparison	29
6.1 Ideal BR-VIPER abstract overview	34
6.2 Used architecture	36
6.3 Landing page of application ...	36
7.1 Possibilities of AR presentation	40
7.2 Augmented reality in Self-acupressure application	43
8.1 Task for the testers	46

Tables

2.1 Table with comparison of downloads and ratings for apps from Google play	6
2.2 Table with comparison of available apps in mobile platform stores.	6
7.1 ARKIT evolution	41
8.1 Self-acupressure testers	45

Chapter 1

Introduction

Traditional western medicine is efficient and well respected in Europe. Still, some people don't trust it and try to avoid it, as they see an over-reliance on drugs and chemistry. With this in mind, many people are taking a more traditional or natural approach to medicine. Examples of this can be seen in the increase of home births or the decline in vaccine rates in population. Although some of these approaches and their benefits are disputable, overusing medicine may lead to drug resistant bacteria [15] therefore people feel justified looking for an alternative way.

Chinese medicine is one of these types of natural remedies that offers a chemical free solution, acupuncture and acupressure are two parts of this tradition. These two treatments can be used for many common medical problems. Acupressure has a lot of potential as it is non-invasive and can be done without the help of another person. Another less known option is reflexive therapy which is claimed to be even more comfortable to undergo than acupressure and also can be used to diagnose problems [13].

1.1 Acupressure

Acupressure is a traditional Chinese medicine which is often labeled as “alternative”. The concept is based on energy flowing through *meridians* in the body. With correct pressure applied to an acupressure point the flow can be unblocked relieving the body of problems like pain, vomiting, or sweating [6] [10].

Acupressure has been labeled as “alternative medicine”, however, it should not be referenced as a scam as there are studies which support its functionality [19]. These studies are sparse, and their findings or selection of participants could be labeled as biased [16]. These studies are typically done with two groups. The first group is treated with correctly applied acupressure points, and the second one is treated using fake ones. Such an approach should limit the influence of the placebo effect. Even though the acupressure might not necessarily help, there are no findings of it being dangerous, so even though it might not help directly, it could help mentally as a placebo.

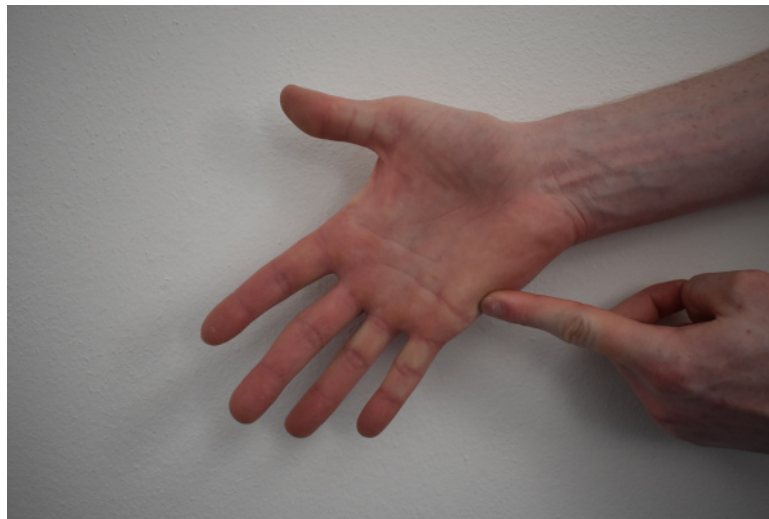


Figure 1.1: One of acupressure points used in application.

■ 1.2 iOS application

Several options exist for someone who wants to practice acupressure. The standard or classical approach is to memorize the points that the person needs to know, or to own a book if that person wants to use acupressure while they travel. The more modern approach is to search for these points on the internet or use a mobile application. As almost everyone carries a mobile phone in their pocket, the modern approach is far more convenient but also has several more advantages.

- It can contain a large number of points with detailed information in small package
- It is possible to search these points to find the desired one quickly. This search can also find acupressure points according to medical problems. Such a search would be a tedious task for a book as one point could belong to many medical problems
- It can be updated quickly, for example, when an illustration image is not descriptive enough.
- It can contain multi-media content like videos or 3d models in virtual or augmented reality.

■ 1.3 Structure

This thesis is structured into 9 chapters. These chapters are following the same order a developer would take when creating an application with one exception. Augmented reality has a chapter of its own, as it profoundly differs from standard application development.

- Chapter 1 is an introduction that contains basic information about acupressure and the motivation behind creating such an application.
- Chapter 2 summarizes and compares existing solutions. Especially for the iOS platform but also for Android phones and web browsers from a mobile user's point of view.
- Chapter 3 contains the specified functionality of the application.
- Chapter 4 informs the reader about developing for iOS, briefly explains commonly used architectures and design patterns, and contains some information about commonly neglected problems.
- Chapter 5 shows the designed application and explains UX choices.
- Chapter 6 describes the implementation of the Self-acupressure application. Architecture which was used for development is explained here. It also contains a description of used libraries.
- Chapter 7 has information about augmented reality, how it could be used for acupressure and implementation details for this module.
- Chapter 8 describes user testing process of Self-acupressure application.
- Chapter 9 is a conclusion with a summary of achieved goals. Furthermore, including a discussion about a possible future for the application.
- The final design of the application can be found at the end in appendix B. Screenshots from the finalized application are in appendix C.



Chapter 2

Existing solutions

This chapter contains a summary and comparison of existing solutions for acupressure. The primary analysis was done on the iOS store, but to gain more perspective, some of the Google Store solutions and solutions on the web were analyzed as well. The iOS and Android applications are often different but basic ideas and flow can be the same as it is the same platform. Web solution was taken into account from the perspective of a mobile user.

The iOS store contains over 30 relevant applications which can be found with search keyword "acupressure". No app seems to have a relevant success as only a few of them are kept up to date. Sadly there is no way to get a number of downloads on the iOS store. Nevertheless, we can somewhat measure the popularity of the application by its position in the ranked list for an app type. By this measurement, most successful is "Acupressure: Heal yourself" as it has reached 33. position in health group. No other app for acupressure has reached a position in such a list.

Another public information about the application is its number of ratings. There is a correlation between a number of downloads and number of ratings. This means we can get a rough estimate for downloads by the number of ratings. This estimate will be imprecise because applications use very different strategies to get their rating. For game application, there can be very aggressive ways to get ratings. Like popups blocking the game. Or a more pleasant way, promising reward after rating application. No such behavior was spotted amongst tested apps, so we can assume only a small amount of users will actually rate the application. This assumption is supported by table 8.1, which show some applications and their relation between downloads and ratings. Number of ratings on Google store is divided into categories with range of downloads. We can that highest percent of rating is little over one percent but usually even lower.

App name	Number of downloads	Number of ratings	Rating percentage
<i>Acupressure Point Tips</i> ¹	50,000+	20	0.04 - 0.02
<i>Acupressure Body Points [YOGA]</i> ²	100,000+	456	0.456 - 0.1
<i>Acupressure Point for 300+ Diseases</i> ³	10,000+	129	1,29 - 0.26
<i>Acupressure Points Guide</i> ⁴	1,000+	8	0,8 - 0.16

Table 2.1: Table with comparison of downloads and ratings for apps from Google play

Even if we estimated that 0.1 percent of the users rates the app, we could assume that the iOS applications have fewer users than in a 1000 as the number of ratings per app varies between one and zero. Following table 2.2 offers quick comparison of selected applications from which some of them will analyze more detailed later in section 2.1.

Name	Platform	Down-loads	Rating	Business model
<i>Acupressure: Heal yourself</i> ⁵	iOS	-	(33 in health)	Paid (50 Kč)
<i>Lite apps (15 mutations)</i> ⁶	iOS	-	(0-2 per app)	Advertisement
<i>Acupressure Points</i> ⁷	iOS	-	-	Advertisement
<i>Acupressure Massage Qigong (YMAA)</i> ⁸	iOS	-	-	Paid videos in app
<i>Acupressure: Self Healing Massage Therapy</i> ⁹	Android	500K	4.2	Advertisement

Table 2.2: Table with comparison of available apps in mobile platform stores.

¹<https://play.google.com/store/apps/details?id=com.tips.acupressurepoints>

²https://play.google.com/store/apps/details?id=com.ERICA_kurse.pressure_points

³<https://play.google.com/store/apps/details?id=atozcure.a21acu>

⁴<https://play.google.com/store/apps/details?id=aksharclearing.agency.acupressurepoints>

Applications on the iOS store can be divided into several groups. The most populated group is a simple app with a list of the points or problems from which the user can navigate to problem detail. This detail consists of point location descriptions and videos or photos of given points. The second smaller group with only two items is an application with a 3d model inside. Another trend or tactic seen in the iOS store is apps labeled with the icon as "lite" these apps utilize the same layout, same or similar content and are only labeled differently to target different audiences of the iOS store. These icon can be seen in 2.1.

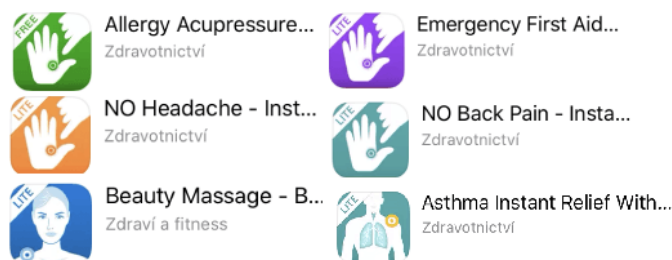


Figure 2.1: Applications using same template with different name to increase its reach

⁵<https://apps.apple.com/us/app/acupressure-heal-yourself/id429680650>

⁶<https://www.apple.com/us/search/acupressure?src=serp>

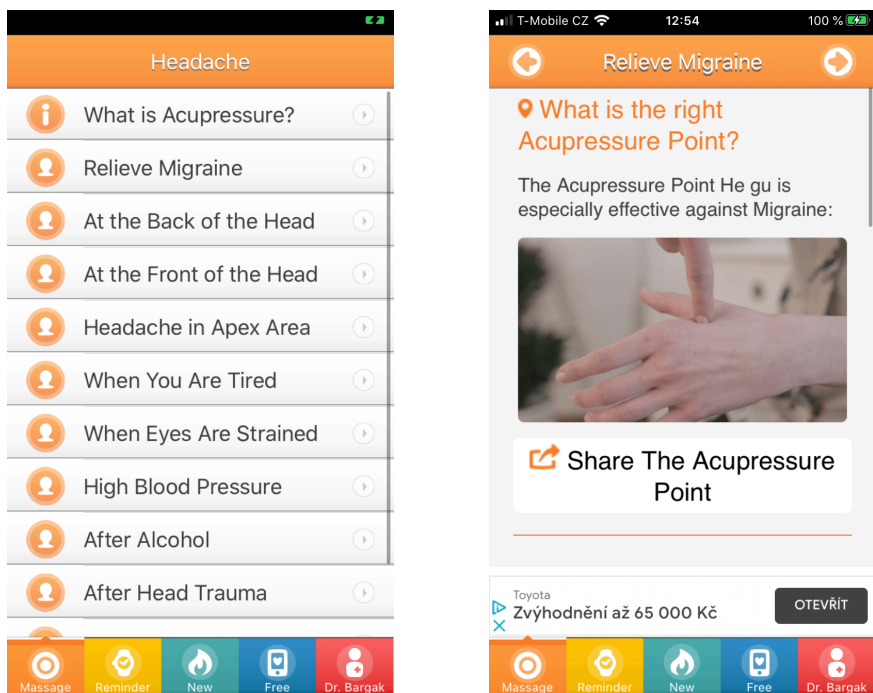
⁷<https://apps.apple.com/us/app/acupressure-points/id1360608057>

⁸<https://apps.apple.com/gb/app/acupressure-massage-qigong/id1293801620?l=cs>

⁹<https://play.google.com/store/apps/details?id=foodanddrink.inc.acupressurepointtips&hl=fr>

2.1.2 Lite apps

This application uses a similar navigation scheme as "Acupressure: Heal yourself", with the bottom navigation bar containing info about app, reminder, and promotion of another app. Except for the landing page, the advertisement is presented at every screen of the application, furthermore advertisement is opened full-screen upon starting the application. The app is not using the native iOS approach and has missing gestures like swipe back for navigation between screens. The app may be considered too colorful, which is distracting. Also, some of the included graphics were not ready for higher resolution phones and are visibly pixelated.



(a) : Main menu screen

(b) : Detail screen of acupressure point

Figure 2.3: "Lite" acupressure application look

2.1.3 Acupressure: Self Healing massage therapy

This application is most downloaded acupressure application on Google Store. Navigation through the app is very inconvenient as there is advertisement popup with every navigation to a child screen. Additionally, the user needs to select a language every time he cold starts the application. Some included icons are visibly in low resolution. The application itself follows the same pattern as others, with the exception at a detail of the point. Points are grouped into categories but are then displayed together in one long detail screen 2.4b.

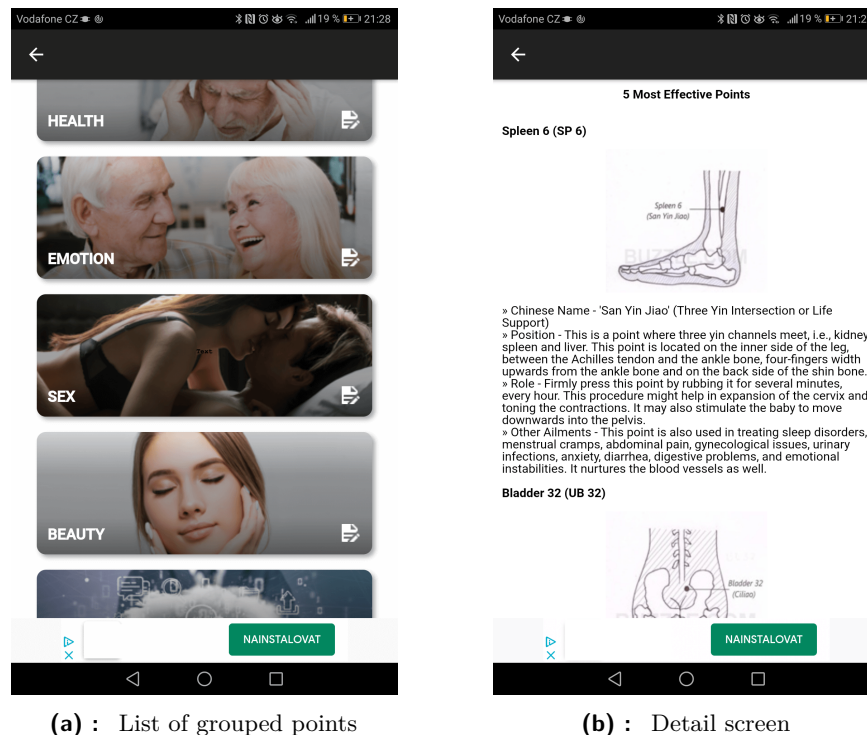


Figure 2.4: Acupressure: Self Healing massage therapy application screen examples

2.2 Conclusion

There is a potential to create an application for acupressure. Existing application popularity is low, with applications having relatively few ratings. Furthermore these applications are not kept up to date. Even the updated apps are not following iOS guidelines patterns, which leads to less intuitive navigation. One of the weaknesses of these applications is also missing good UX/Desing patterns. With such an analysis, we have information on what to use and what to avoid.

Chapter 3

Application functionality

This chapter contains list of requirements for the application. Some of these requirements were derived from the comparison of book and an application. The stated advantages should be implemented to use the whole potential of a mobile platform. Another set of requirements is derived from previous chapter, these requirements from existing application so their inconveniences would not be repeated.

3.1 Functional requirements

Functional requirements are functionalities that developer must implement to enable user of the application to accomplish their task.

- **FR1: Viewing acupressure point**
The application allows user to view acupressure point detail with information about point function and location.
- **FR2: Searching for point by problem**
The application groups points by problems.
- **FR3: Searching for point by its location**
The points are grouped by location in application.
- **FR4: Viewing acupressure point in augmented reality**
Acupressure point can be inspected in augmented reality.
- **FR5: Liking point**
The application allow user to like point.
- **FR6: Data deletion**
It is possible to delete all stored data.

■ 3.2 Non-functional requirements

Non-functional requirements or quality attributes which describes general characteristics of the application.

- **NFR1: Deployment target**
Application will be created for iOS mobile devices.
- **NFR2: System version**
The application should run on iOS of version 12 and higher.
- **NFR3: Language**
The application should be in Czech and English language.
- **NFR4: Persistence**
The application should persist important data.
- **NFR5: Dark mode**
The should application allows change of view mode.
- **NFR6: Design**
The should application follow iOS design guidelines.

■ 3.3 Use cases

Use cases defines interaction of a user with the system. These interaction are meant to be a typical workflow of a user.

- **UC1: First start of the app**
 - The user opens application.
 - The user see on-boarding and is able to continue to main screen.
 - The user is on the main screen.

- **UC2: Finding acupressure point**
 - The user opens application.
 - The user chooses his problem from list with problems.
 - The user see acupressure point detail.

- **UC3: Viewing acupressure point in augmented reality**
 - The user opens application.
 - The user chooses his problem from list with problems.
 - The user see acupressure point detail.
 - The user is able to view point.

- **UC4: Finding all points on hand**
 - The user opens application.
 - The user taps to choose hand from locations list.
 - The user sees list of points on hand.

- **UC5: Liking a point**
 - The user opens application.
 - The user navigates to a point detail.
 - The user likes the point.

Chapter 4

Development for iOS

This chapter will explain some basics of development for iOS from language choice to the architecture pattern.

4.1 Language choice

When choosing a language for iOS, the developer has two choices if he wants to develop a native application. The first choice is objective C, which was used since 2007, when the app store was introduced. Objective C is fast and powerful but it perhaps gives too much of a choice to the developer when handling memory, for example. Swift was introduced in 2014, and it was a massive surprise for the community. From then, SWIFT has taken the lead as the primary language when developing iOS applications. SWIFT is now the only fully supported language as the C is often omitted when creating tutorials for new features.

Swift and C are not only options when developing an app for iOS. Other options mainly aim to create one code base for both iOS and Android. A single code base comes at the price that the application is missing the native user interface. Standard multi-platform options are Flutter, react-Native, or PWA.

4.2 Swift

SWIFT is a modern programming language. It is strongly typed and offers nil handling. Development is done in a free tool named Xcode, which is available only for Mac iOS. The starting point of the application is App delegate. App delegate serves to start the application and as a mediator for notifications and deep-linking.

Another integral part of an iOS application is the *Info.plist* file, which contains important information about application. This information is, for example, requested user permissions, used fonts or version. This file needs to be localized as it also contains a description of how will be the permitted parts of

the system will be used. This description should contain all uses of permission; otherwise, it is at risk of being removed from the App Store.

■ 4.2.1 Design patterns

■ Navigation

`UINavigationController` is basic component used for navigation between screens in iOS. This navigation controller is associated with a root screen and contains info about the navigation bar. When we want to present another screen, we can present it modally, which means like a popup screen, which can be dismissed by swiping down gesture. Another option is pushing the screen to the stack. This screen would belong to the same navigation controller and would have the same navigation bar. It is possible to place a screen on the stack almost indefinitely, but the programmer has to keep in mind that screens on stack still occupies space in memory. Navigating back in the stack is called popping. It is possible to pop to the previous screen or to the root screen. It is also possible to push a new navigation controller to the stack and should be done this way when presenting a new standalone part of the application.

■ 4.2.2 Arc

Automatic reference counting abbreviated ARC is a memory management system used for the SWIFT language. Each allocated object is counting a number of times it is referenced. Increasing the number when it has been referenced and decreasing when it has been dereferenced. The object exists in memory while the number of references is greater than one. When not kept in mind, the problem of retention cycles can occur. This happens when two or more objects create a cycle of references preventing them from being dis-allocated. Such a cycle can be seen on 4.1.

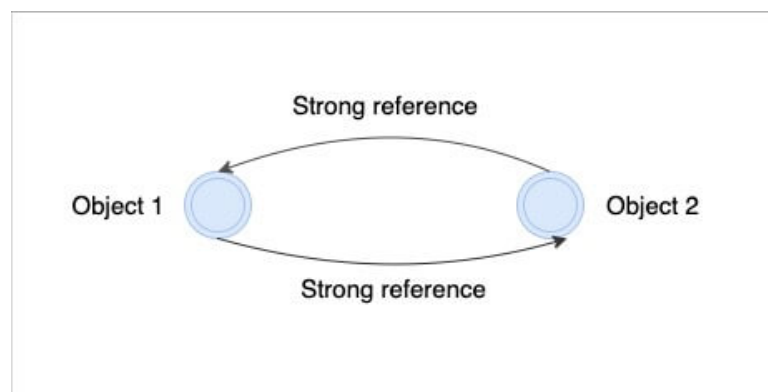


Figure 4.1: Simple retention cycle

This cycle can be broken with keyword **weak** or **unowned**. References with this keyword will not increase the reference count. Objects with this keyword

may not exist if they were de-allocated. More complex problem of retention can be seen in 4.2. This reference cycle also prevents all pattern layers above from being dereferenced. This retention would lead to memory problems as all of these objects would be created each time this screen would be opened, leading to a forced crash of application by the system if it would waste too much of memory. This problem should be fixed with **weak** keyword. When fixed, all layers of architecture would be freed from memory when the pointer for view controller ceases to exist, this typically happens when the user is navigating back to the child view controller.

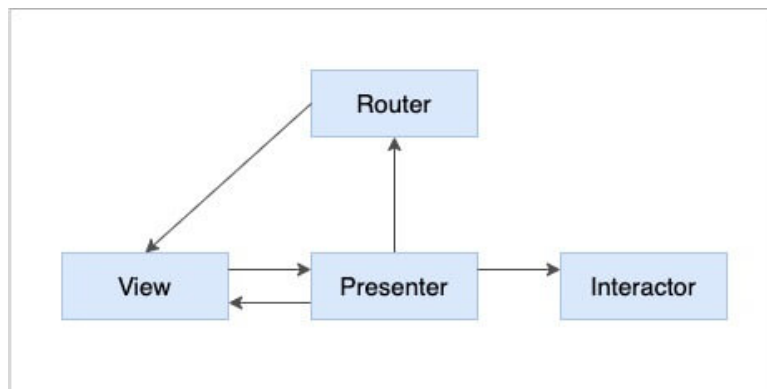


Figure 4.2: Retention cycle in VIPER

Checking the correct implantation of memory management can be done with Xcode built-in tools. A fast check can be done with a memory report. We should see spikes when navigation back and forth on the screen as the memory is being allocated and freed. If the memory is only increasing over time, the retention cycle is present and should be removed, to avoid crashes and wasting precious memory.

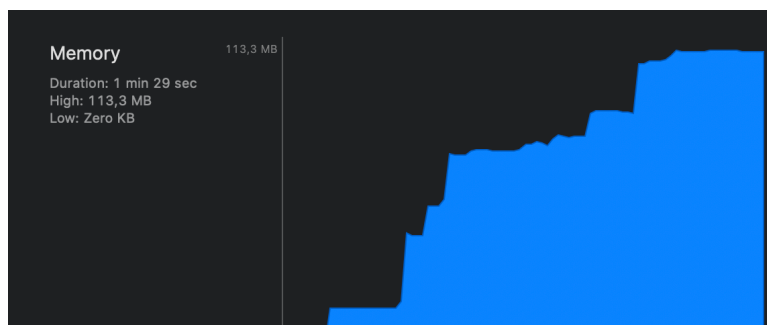


Figure 4.3: Rising memory indicating possible memory problem

Another option is to get a list of allocated objects with a profiling tool named Allocations. When navigation back and forth on the screen, we should see objects being created and then freed from memory. We can also observe the total number of allocations and current allocations. The number of total allocations should be typically smaller than the number of current allocations.

■ 4.2.3 Libraries

Libraries for iOS are mainly distributed by CocoaPods, Carthage, and native Swift Package Manager. Of course, the programmer can download code directly from some source control, but this comes without automated checks of dependencies and updates.

■ CocoaPods

Open source dependency manager. It has many advantages as it is easy to use. Almost every framework is uploaded there. It is simple to find dependencies the application uses, and it can find conflicting dependencies. The main disadvantage is when building project, all dependencies will be built as well [9].

■ Carthage

Carthage supports finding conflicts and dependencies, as well. It is decentralized, which means there is no single point of failure, but its harder to find new libraries. Also, not every framework is available for Carthage. The ability not to build every dependency comes at a price of too many steps when adding a new dependency.

■ Swift Package Manager

SPM is a new standard way how to create libraries. Same as CocoaPods and Carthage, it automatically checks dependency's dependencies. It has direct support from Xcode and does not need to be handled from the terminal. Not all frameworks are here, but some big projects like Stevia are currently dropping support of Cocoapod in favor of SPM.

■ 4.2.4 Distribution

Swift is built similarly to objective C. Code is built not interpreted. This means that Swift, unlike Android, does not need obfuscation. The built file is in *.ipa* format and contains builds for all enabled architectures. This file can not be directly distributed as all application needs to be signed. This means that the application can be distributed to consumers only via AppStore. Testing of the application can be done on a real device if distributed with Testflight, or the device has to be added to the signing certificate.

When creating an application, the whole process is normally as follows.

- Build application.
- Upload IPA file into app store connect via loader.
- Apple automatically test the uploaded file. These tests checks, whether permission are presented, finds deprecated API usages, and more.
- Distribute the application for testers (optional).
- Add basic store information. Screenshots from application, version, and description text.
- Submit for review. This review is done by Apple review team and checks whether the application serves purpose its supposed to serve, runtime errors, and other problems. This usually takes between 2-5 days.
- Fix reported issues from review team.
- Release the application.

■ 4.2.5 Architecture Patterns

Without using the architecture pattern, the View controller quickly becomes too large to be handled efficiently. Architecture patterns should not be confused with application architecture. The architecture patterns are mainly for screen level code handling and without higher abstraction will probably be full of repeated code.

■ MVC

MVC stands for model-view-controller. It is a default iOS pattern. As seen on image 4.4 view receives updates from controller. Controller updates view and model and is notified about changes in the model. This means that the controller is handling all screen logic, navigation, and communication with the model. This leads to a problem where the controller part is large and hard to break apart. There is a joke which says that MVC actually stands for the massive view controller.

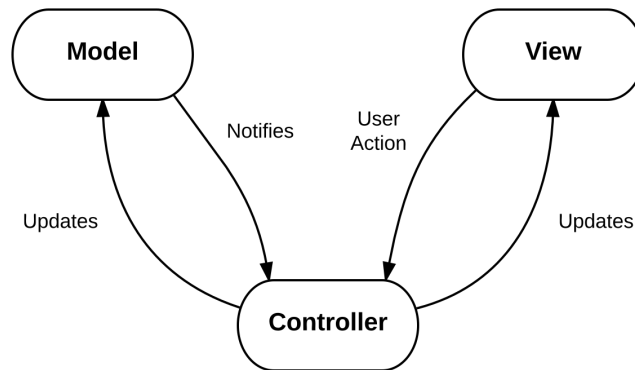


Figure 4.4: MVC architecture pattern [2]

■ MVVM

MVVM stands for Model - View - ViewModel. This pattern can be marked as an augmented version of MVC architecture. In this architecture, the view part is an iOS View controller. View controller no longer handles action and screen logic but sends interactions to View Model. View Model handles presentation logic, updates model which handles business logic and data. This architecture works well with reactive programming, as there is only one-way binding.

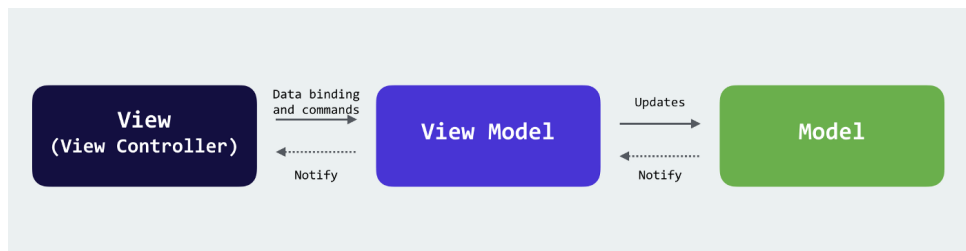


Figure 4.5: MVVM architecture pattern [18]

VIPER

Viper is an adaption of Clean architecture for iOS. Viper is an abbreviation for **V**iew, **I**nteractor, **P**resenter, **E**ntity, and **R**outer. VIPER architecture overview can be seen in image 4.6. With five components, it most extensive of used pattern and requires lots of boilerplate code. In image 4.6, full lines mean that the object is owned in terms of ARC. Dotted lines mean only weak reference is created. This means when the view is dereferenced, all of this structure is dereferenced as well. This pattern can be further augmented with reactive patterns [17]. This means that only strong references remain as the reference from view from **P**resenter to **V**iew is no longer needed. Furthermore reference from **I**nteractor to **P**resenter is redundant.

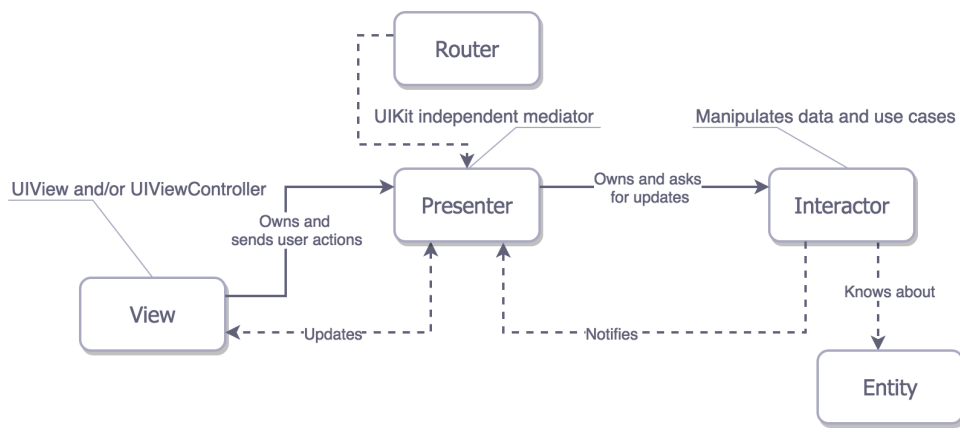


Figure 4.6: VIPER architecture pattern [8]

View is UIViewController, and it tells the **Presenter** about user actions and displays content based on updates from the **Interactor**. **Presenter** contains view logic and state of the view. It also handles user action received from view. It asks **Interactor** for updates that it receives. It also asks the router to do navigation. The **Interactor** contains business logic. The **Interactor** and presenter should work on a use-case basis. The **Entity** is name for basic models that **Interactor** uses. **Router** controls application navigation.

Sometimes this pattern is bent to be more efficient, so **Entities** and **presenters** are not created on use case base but per screen. Lastly, displaying alerts is often not propagated to the presenter and then router but often handled on lower levels as it is more convinient.

Chapter 5

Design and UX

This chapter contains a proposed design for Self-acupressure application. The proposed design and final application differ as some of the proposed functionalities were redundant. Also, the user testing discovered some possible improvements which were immediately incorporated into the application. The app will be for iOS only and written in SWIFT, so the design guide that was used is adhering to iOS guidelines [12]. These guidelines are based on six principles.

- Aesthetic Integrity - How well the app appearance and its purpose/behavior go together.
- Consistency - The app incorporates standard system elements.
- Direct Manipulation - User manipulation immediately affects the content.
- Feedback - The user has feedback on his action. For example, interactive elements are highlighted when tapped, or he can see progress indicators when downloading new data.
- Metaphors - Using actions that are familiar to the user.
- User control - Users should be in control. All actions should be predictable to avoid unwanted outcomes.

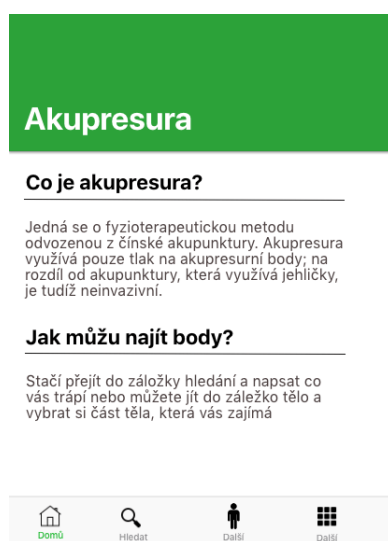
These guidelines were taken into account when designing the Self-acupressure application. To create the design Adobe XD ¹ was used as it is intuitive, offers creation of live prototypes, and allows to download of native components that can be used as a fundamental building block for application. These basic blocks should be used to follow the consistency principle of iOS guidelines. These blocks can also significantly increase the speed of development. A live prototype can contain the whole flow of the application. This can be used to find problem in UX and design quickly, saving precious development time. This design was created simultaneously with design for Android application, which is a part of a different thesis [14].

¹<https://www.adobe.com/cz/products/xd.html>

5.1 Navigation and flow

One of the most challenging things when developing an application is navigation. The application should be intuitive, with non-disturbing navigation. Also, with the use cases in mind, the typical actions of the user should be done fast with minimum effort, which means having as few taps to do the action as possible.

For Self acupressure application, the bottom navigation bar was incorporated. This navigation bar is visible from the main page and from its other pages, which allow users to see where he currently is and to where he can navigate. When comparing the bottom navigation bar to the side navigation bar and navigation bar, several advantages for the bottom bar appear. As said, the bottom navigation bar is always visible, which also can be a disadvantage because it takes screen space permanently. Another advantage is that it is in an always accessible area for the user and simple tap with his thumb take him to the wanted page of the list. For the side navigation and top navigation first tap is needed to see the menu. The second tap is needed to navigate to the chosen item; also, these items are probably in the upper part of the screen forcing the user to hold the phone differently or use a second hand.



(a) : Bottom navigation bar



(b) : Navigation from detail

Figure 5.1: Navigation in self acupressure application

The navigation menu will not be used because it is not an intuitive solution for the user. The navigation menu should contain more advanced actions. The side menu has an advantage as it can show information about the user and his profile, but in this application, no profile will be created, and it is also not a native part of the system.

In figure 5.1a, there is a visible bottom navigation menu with four menu items. The home which serves as a landing page for the application. Search page which contains list of acupressure points that are possible to be searched. Body screen which has points grouped by their position. On top of the screen in figure 5.1a, we can see a navigation bar with a large title, which is a feature introduced in iOS 11. Large titles were used as they give application identity because they take space with the theme color of the application. At the same time, when used for the screen with the scrollable component, they behave sticky, so when the user scrolls up, the large navigation bar reduces its height. When in detail like 5.1b the top navigation bar can stay large or is reduced based on the screen content. In both cases in the top left corner, there is a back button for navigation to the previous screen. To speedup the navigation swipe back gesture is present on each of these detail screens.

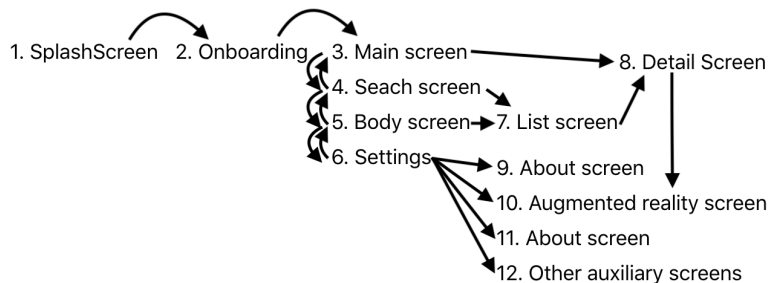


Figure 5.2: Navigation flow of Self-acupressure application

The basic flow of the application is displayed in figure 5.2. Screen 2. Onboarding is displayed only once to show the user basics of the application. Screens 3,4,5,6 refer to bottom navigation screens. With the exception of screen 6. all these screen can be used to navigate to detail screen as it is the most crucial screen of Self-acupressure application. Screen 6. setting contains augmented reality for viewing acupressure points, about screen with information about the app, contact screen to report bugs, and other setting screens.

5.2 Design

This section shows the design of the most crucial screen of Self-acupressure application. The rest of the design can be found in appendix B.

5.2.1 Onboarding

Onboarding is a vital part of many applications. Its purpose is to show the user what to expect from the application and/or how to use some more complicated parts of the application. The onboarding should be fast; otherwise, the user could be discouraged by it. Onboarding for Self-acupressure was designed that the user can swipe through its pages similar to a book. It is also possible to tap the *continue* button to navigate to the next screen. *Skip* button allows

users to skip onboarding entirely as the user might have seen it already before. The dots in the top part describe the user position in this tutorial. The first choice was to explain to the user how to like a point, how to find points with search function, and that the tapping on the body, as seen in figure 5.6b, leads to a list with points located on the corresponding body part.

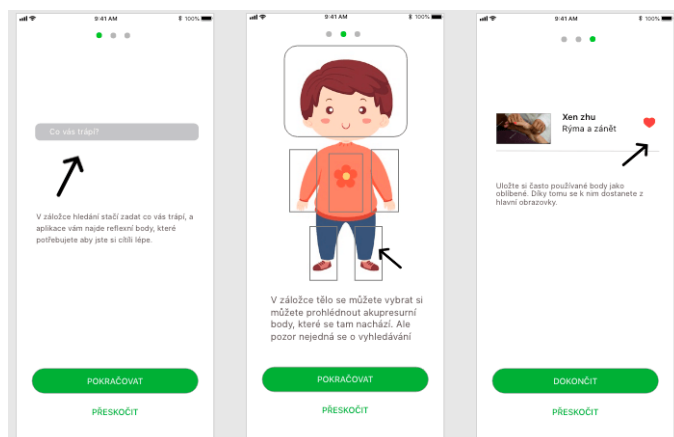


Figure 5.3: Self acupressure onboarding design

5.2.2 Main screen

The landing page is the first screen the user sees when he typically opens the application. The screen has two states. First is visible when the user has no liked points 6.3a. The screen in this state shows necessary pieces of information about application and acupressure. Second state 6.3b is shown when the user has at least one liked point. A list of favorite points is shown in this state. User can also dislike previously liked point by tapping the like button which would remove the point from the favourite points list.

5.2.3 Search screen

The search screen has a table of every acupressure point. In the top navigation bar, there is a field for searching. When the user writes his problem into this field, the list is automatically updated with relevant results. This screen was omitted from the final implementation for more information read the next section comparing design and created application. 5.3

5.2.4 Settings screen

The setting screen consists of one scrollable table grouped into three categories for faster navigation. It consists of navigation to augmented reality, app settings, information about the application, and author.

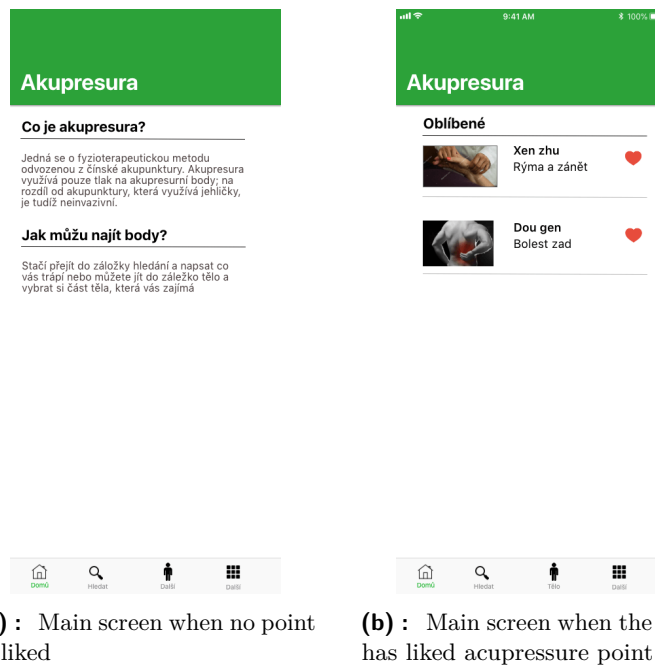


Figure 5.4: Landing page of application

5.2.5 Detail screen

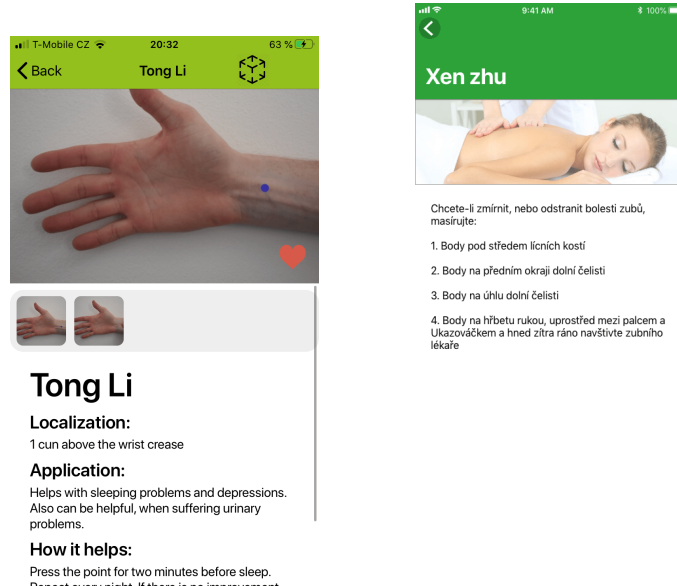
The detail screen was designed very simple, with large picture on the top of the screen so the user has visual localization of the point. This picture could be enlarged even more with a tap. Moreover, under the picture is located text about how to use the acupresure point and where it is located. This screen was changed heavily after feedback from the first testers. These changes are explained in next section.

5.3 Design and implementation differences

The most significant differences between final application and flow are different items in settings, quicker navigation to augmented reality, and search screen, which been changed to problem screens. The following section compares these differences and explains why they were made. For full design and implemented application and how it looks on different devices, go to the appendix C and D.

5.3.1 Detail screen changes

The detail screen was augmented. In the top right corner, there is a navigation button to augmented reality screen. This simplifies movements in the app as previously user had to go back to setting screen and then to augmented reality. This also makes it possible to zoom on a given point in the AR screen.



(a) : Detail screen on iPhone 8 (b) : Proposed design of detail screen

Figure 5.5: Landing page of the application

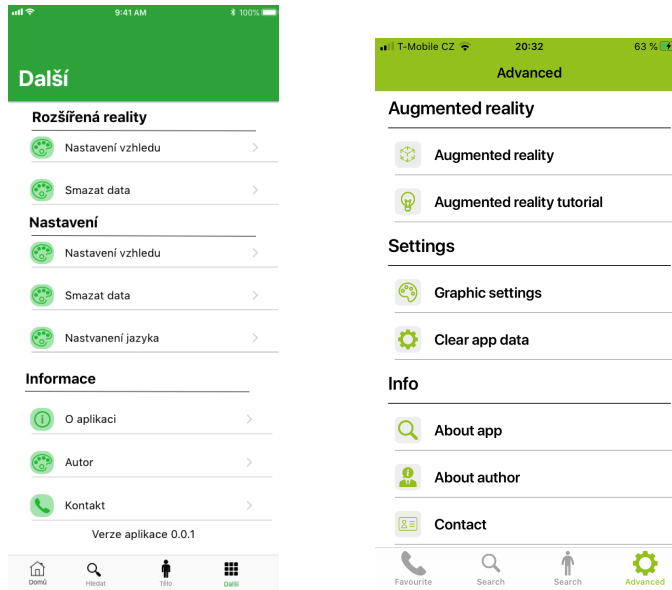
Another change on this screen is added support for multiple images of acupressure points, which should improve localisation and visually show how to press the point. Furthermore, different text layout was added which now also explains how to use the acupressure point.

5.3.2 Settings screen changes

The language setting was removed from this screen as the application automatically changes its language based on the setting of the phone. Also, the graphic settings only contains hint on how to change the dark mode and could be theoretically removed.

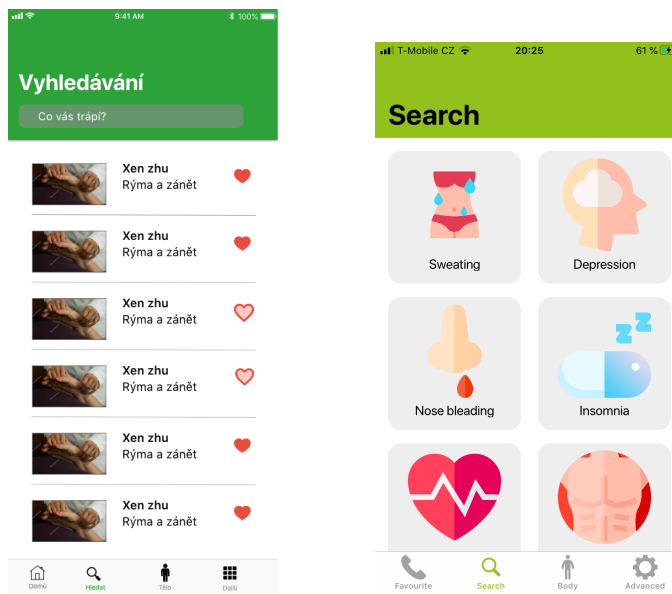
5.3.3 Search screen

This screen now consists of a collection of cards-like views that contain grouped points by problems as visible in figure 5.7b. The search functionality was removed as the number of problems is relatively low, so it is not difficult to navigate in them.



(a) : Proposed design of settings screen (b) : Settings screen on iPhone 8

Figure 5.6: Settings screen design and implementation comparison



(a) : Proposed design of the search screen (b) : Search screen on iPhone 8

Figure 5.7: Search screen design and implementation comparison

Chapter 6

Implementation

6.1 Cocoa pod usage

Several pods were used when developing Self-acupressure application. The following list is a quick summary of used pods. Most interesting pods are described in detail in the following subsections.

- **R.swift**
 - Management of resources like images and strings. This simplifies their loading.
- **SteviaLayout**
 - This pod allows the programmer to create visual auto-layout constraints in code.
- **RxSwift**
 - This pod adds reactive pattern to SWIFT.
- **RxDataSources**
 - This pod allows the binding of data and collections together. This simplifies the handling of complex data that is changeable.
- **Kingfisher**
 - This pod is taking care of downloading and caching images.
- **Hero**
 - This pod is used to create between-screen animations.
- **SynBase**
 - This pod used to manage the key-chain and user defaults.
- **CollieGallery**
 - This pod is used to show the detail of the image. This detail is zoomable, and the user can also share the picture from there.
- **Moya**
 - Moya is a pod for handling API requests.
- **WeScan**
 - This pod is used to scan images for augmented reality.

6.1.3 RxSwift

This pod brings reactive programming for SWIFT via an Observable interface. This is very useful as there are counterparts for this paradigm in every major programming language. This can simplify on-boarding and adds the possibility to share architectures and ideas between different platforms.

When writing with RX the created code can be easily made functional as the created stream has clear inputs and outputs. It is also more natural not to create overly-complicated state-full pieces of code. For front-end development, it can observe UI events, which simplifies data passing between layers of architecture. Also, thinking about code can become cleaner as everything in RX is treated asynchronously.

```
request(.getAllPoints(language: code)).do(onSuccess: { [weak self] response in
    print(response)
  }, onError: { [weak self] error in
    print(error)
  })
.map(AppDataEntity.self)
.subscribe(onSuccess: { [weak self] model in
    print("Data downloaded succesfully")
  }, onError: { [weak self] error in
    print("error loading data")
  })
```

Listing 6.5: Handling request with RX

Another powerfull thing about RX is that the programmer has all information on the end of the stream, which he needs. Single means this event will be triggered only once. Observable means numerous events can happen. The driver pattern is use-full for handling UI as it has no error and requires default value to be passed when an error occurs. Rx can also be used to enhance VIPER pattern [17]. Usually, both layers of architecture have to know about each other. With RX, only the lower layer knows about the upper. The lower layer informs the upper layer about actions and subscribes information it needs.

6.2 BR-VIPER

For self-acupressure, the VIPER pattern was heavily modified. Two new layers were added. **Builder**, which is per screen layer and takes the responsibility of router to create VIPER modules. **Repository**, which handles per feature business logic. Also, all dependencies are stored in one class to allow for easy dependency passing. The interactor now serves only as a proxy to repositories and can map or aggregate informations together; it also contains dependencies and can serve information from them to the presenter when asked. Nevertheless, it often serves only as a proxy from which we quickly see how the screen is communicating with the repositories. The main difference

between repository and dependency is that the repository uses dependencies and is created per feature. Dependency is created per functionality like network communication, database storage, or key-chain access. Component of viper are loosely coupled via protocols. This increases their testability and intractability.

To speed up the development process, every screens basic VIPER module was created automatically. This required to have all architecture layers communication known before implementation. Automatically generated boilerplates can save about 30 minutes per screen and means that every screen module will be written in a similar way, which lower misunderstandings between multiple programmers.

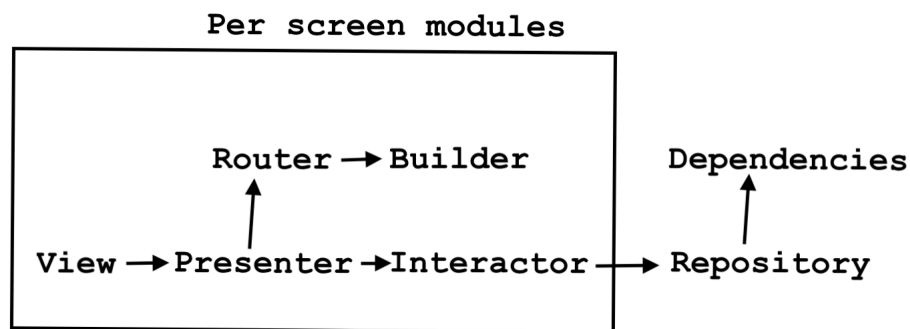


Figure 6.1: Ideal BR-VIPER abstract overview

- **View**

Consist of a View controller. Handles the layout of views and has delegated actions from tables. All actions from used views are passing through here to the presenter. It owns the presenter and observes its data and updates its layout or its children's layout by them.

- **Presenter**

The presenter owns an interactor and router. Deals with screen logic and keep its state. It has no information about the view and does not directly update it.

- **Interactor**

The interactor is implemented as a struct. It gets data from repositories and is also stateless. It contains dependencies and updates them or get a data from them.

- **Router**

It can show another screen when told by the presenter. It should also show alerts. It asks the builder to create a screen with which it is associated. Router also tells other routers to show their screen.

- **Repository** Can take care of caching. Handles business logic of one feature. Gets or saves data to/from dependencies.

■ 6.3 Architecture overview

■ 6.3.1 Dependencies

Self-acupressure app is using following dependencies.

- `AcuPointsProvider`
- `KeychainInteractor`
- `UserDefaultsInteractor`

`AcuPointsProvider` is taking care of networking. `KeychainInteractor` is handling interaction with keychain. Keychain allows storing simple values; these values, if the user has enabled iCloud, are distributed amongst his devices. `UserDefaultsInteractor` is used to save information about app progress. For Self-acupressure it stores information if onboarding and AR onboarding were shown.

■ 6.3.2 Repositories

Repositories handles features and provides functions to screen interactors to avoid code repetition.

- `ArRepository`
- `AcuPointsRepository`

`ArRepository` contains information about available models, and points that can be displayed in AR. `AcuPointsRepository` contains business logic for acupressure points. Through interface it allows other part to reactively get points or points for given part of body. It is also handling caching of downloaded acupressure points with fallback to file containing basic acupressure points.

■ 6.3.3 Screen modules

All these screen modules are created with full VIPER structure. Screens are written without Storyboard, due to usage of Stevia pod.

■ 6.4 Dark mode

The dark mode is one of the often requested features for apps and IDEs. This feature has been introduced for iOS 13. An application like Facebook messenger had implemented this functionality even before it was natively

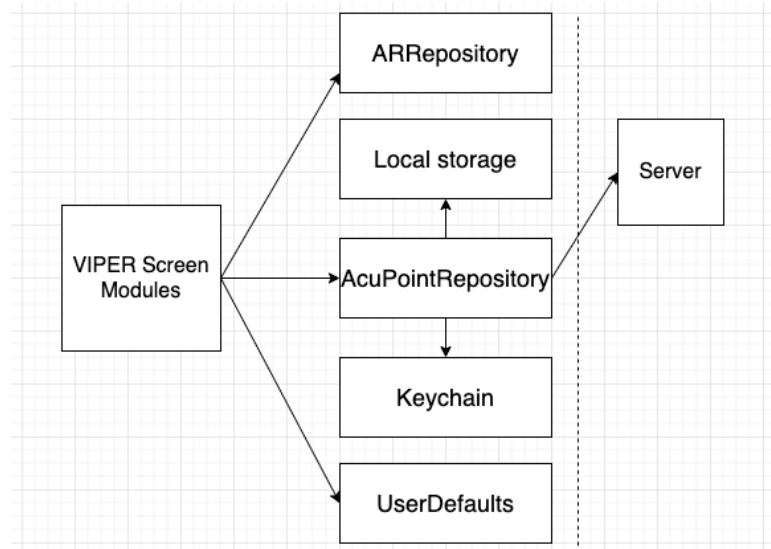


Figure 6.2: Used architecture

supported, and Android has not got these features up to date. This feature has become so popular because phones become an integral part of our lives, and we used it everywhere and always. This includes night time and before going to bed. Using dark mode reduces exhaustion of eyes and looks better with a blue light reduction. Even though it is not significant dark mode can save energy [3].

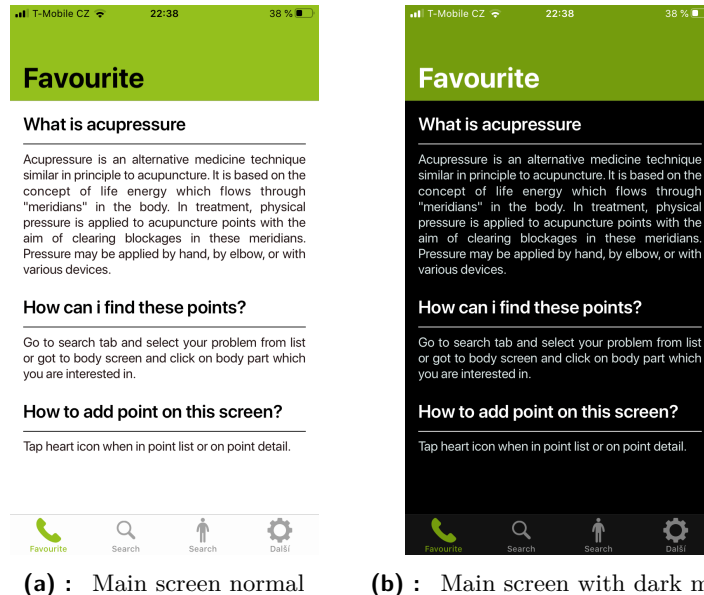


Figure 6.3: Landing page of application

6.5 Resources

Creating a mobile application requires lots of resources. For Self-acupressure, these resources are images, fonts, texts, colors, and augmented reality models. These resources should be handled well, so they can be reusable, easy to find, and use.

6.5.1 Localisation

Resources for the iOS application can be localized. This mainly applies to text and images. This localization is done directly in Xcode, and no changes in code are needed. Text localization of acupressure points is done on the server-side, and the application only sends language code with its API calls. Static string in the application is localized within the app. This file contains key and value for every needed string. The key is used to reference string from the application. However this format is not convenient for translations. To simplify the work needed an online translation tool named Loco ¹ was used. This tool makes translation easier as the unlocalized strings are visible and when localizing the source language string is visible as well. This tool also offers easy exporting in many formats that came handy when localizing the Self-acupressure Android project as these projects use the same strings and keys, lowering the amount of time needed when deploying new or fixed localization.

6.5.2 Images

Images used in the application are in png and pdf format. Pdf format is preferred as it is a supported vector format for iOS. Vector images are smaller and can be scaled indefinitely, meaning they won't be obsolete when a new screen with higher resolution comes to the market. Non-vector images should be divided into three categories of resolutions to support different screen sizes. If these categories are supplied with only high resolutions photos, the final application could be unnecessarily big. When provided with low resolution, the application images will be pixellated for higher resolution screens.

¹<https://localise.biz/dashboard>

Chapter 7

Augmented reality

Augmented reality is slowly becoming closer to be used on an everyday basis. It could be viewed as the next step in product marketing. Similar to the advances from text to images or from images to videos. Augmented reality has many possible usages, such as displaying items in an interactive way, indoor navigation or playing games. Another usage could be merging reality in a more immersive way for example adding furniture to an existing room.

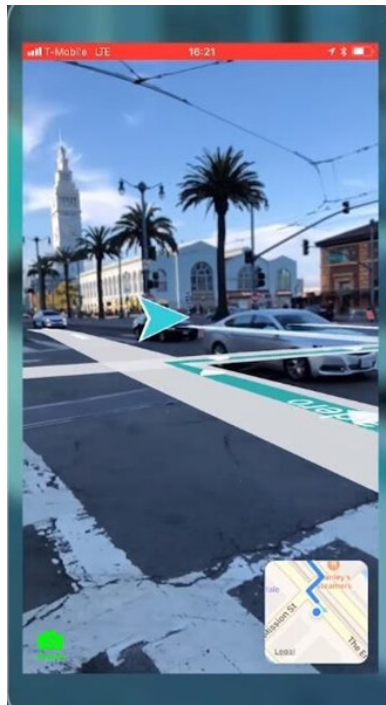
7.1 Basics

Experience with augmented reality could be divided into three categories for purposes of this application. These categories are based on how much the displayed content has to know about reality. The first category is a tabletop presentation. With a tabletop presentation, the scene is added to the horizontal plane. The added scene can be moved around, scaled, and be interacted with. It does not have to be necessarily a horizontal plane but could be any point in space. Such a presentation does not require almost any info about its surroundings. This is well suited for showing merchandise or presenting some games.

Second category is augmented presentation. This category can be used when enhancing reality. This means the detection of more advanced structures, such as images or objects. This can be used to alter the images in the frame, or maybe display additional information about them.

Third category is the World scale or World presentation. The content of this category should aim for the seamless integration of virtual objects and reality. This can be used, for example, to navigate through a shopping mall, add new furniture to the living room or play a game where players can interact with virtual objects.

This division into the categories is to aid thinking about AR in more conceptual way and its possibilities. There is no strict border between them, and one AR application can use all these styles to deliver content to the user.



(a) : World scale [4]



(b) : Table top [1]

Figure 7.1: Possibilities of AR presentation

7.2 AR and acupressure

This section contains info on how augmented reality could be used to help with the acupressure. Even though that the AR is evolving quickly not all possibilities here are currently creatable with present technology provided by the Apple.

■ Presenting model

The most simple approach, which display whole body or part of interest in augmented reality as tabletop.

■ Augmenting view

This approach would show the model to the user on detected image or frame.

■ Model integration

This means when the user would point his camera to his hand, hand from object would be placed on top of it. This object would mimic movements of the real hand.

■ Points integration

With this approach, only the acupressure points would be displayed on target. This approach would be far better than the other as the user would see the points exactly on his hand.

All of these approaches would have interactive acupressure points. When such a point would be tapped, detailed information should appear. This information could appear like a new view controller or as a new layer over the AR screen, but not to disrupt the AR experience this information should be displayed while still in augmented reality.

7.3 Frameworks

To create augmented reality for iOS applications, programmers can choose from native and third-party solutions. Third-party solutions have an advantage as they can be created outside the mobile environment and then only incorporated in the mobile application. This means when using Unity ¹, or Vuforia ², one code base for AR can exist for iOS, Android, and Windows mobile, for example. This comes at the possibly difficult incorporation into the project.

ARKit is a native solution for augmented reality on Apple iPhones. Table 2.2 offers summary of version for ARKit. A higher version of ARKit is supported only on newer iPhones or iPads as these technologies rely on specific hardware.

Version	Release	Features
<i>ARkit</i>	June 2017	Object placement, and detection
<i>ARkit 2</i>	June 2018	Shared experience, persistence tracking
<i>ARkit 3</i>	June 2019	People Occlusion, Motion capture
<i>ARkit 3.5</i>	March 2020	Lidar sensor support

Table 7.1: ARKIT evolution

ARKit 2 brings to augmented reality shared experience which allows simple creation of multiplayer games or just sharing current AR experience. Persistence tracking allows saving of current state to share it or reload it later [7] [11]. ARKIT 3 aimed to create AR scene to look as real as possible. To do so, people occlusion was introduced [5], this means that before the AR object was on top of everything nevertheless the distance. Newer phones can now hide object behind people which helps to create feeling that the object is in reality. Motion capture allows to track human as simple skeleton with points on their joints. ARKit 3.5 further increases potential of AR, as it is trying for higher understanding of scene meaning that previous feature should work better.

ARkit was using Scene-Kit to create scenes for AR. This has changed with ARKit 3 as new solution named Reality composer exist. Reality composer is standalone software which can create scenes, simple logic and animations for the AR.

¹<https://unity.com/products>

²<https://developer.vuforia.com/>

7.4 Implementation

For the Self-acupressure Scene-Kit was used to create AR scenes as the Reality composer is supported only for iOS 13 and higher. In current version it is possible to display a hand with all acupressure points presented in application. Two possibilities how to show object in AR were implemented.

7.4.1 Plane detection

First is a plane detection. At first user has to move phone around a little to detect a plane, on the detected plane the hand object is displayed. This requires to have the horizontal plane detection enabled. This options is really convenient for the user as it requires no additional setup and finding plane is fast when in environment with enough light.

7.4.2 Image detection

This options requires additional setup as the user has to scan two images which will be then detected. Ideal images are front and back side if some card. Image scanning is done automatically with frame detection, so the scanned image is immediately cropped. When the image is detected front or back of the hand is showed in AR. The image is tracked live, so the image is rotating with hand, or seen from the other side by flipping the card.

7.4.3 User interaction

The displayed model can be rotated with standard rotate gesture. This model can also be scaled with the pinch gesture. This scaling is limited to avoid problems when the user scaled object too much and would not be able find it. In upper right corner is the advanced actions. These actions allows user to switch between plane and image detection. Last action allows user to restart current AR session. Furthermore tapping on acupressure point which is depicted as green or blue sphere shows acupressure point detail. This detail can be hidden when tapping on other acupressure point and is always rotated so is the text on it visible to the user.

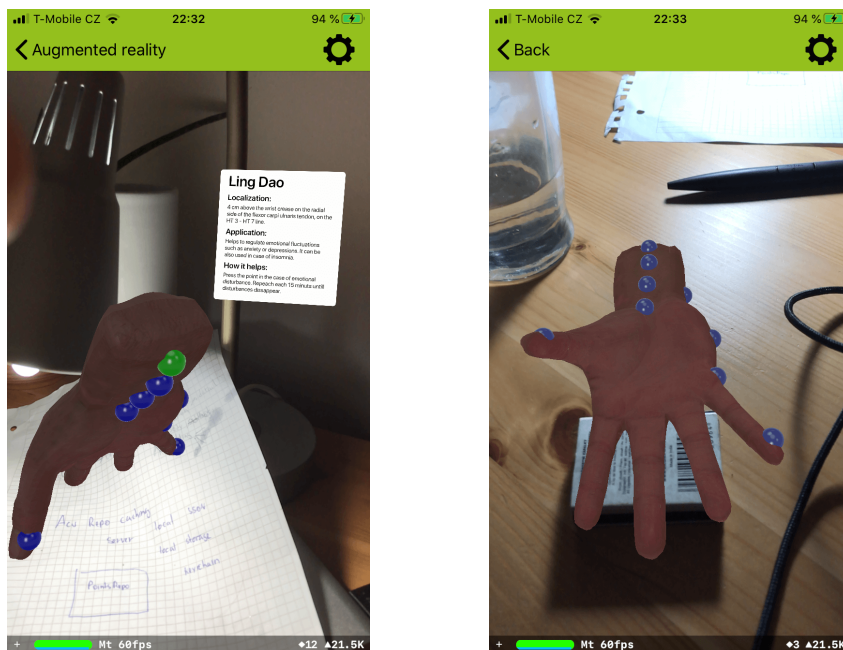
Handling tapping

Gesture recognizer was used when determining if the user has tapped on an acupressure point node. The location of this tap was used to determine if an object was hit in the scene with a ray-cast. If it was the hited an object with a prefix of acupressure point, the tap is successful, and detail is shown. This was unsatisfying when the object was far away from the screen because the tap was often recognized as hitting hand instead of point even though the user has a feeling he tapped the point. To determine tap, a more user-friendly ray-cast could be replaced with a cone or cylinder, however this solution would give false positive clicks. To determined the possible tapped

points, another solution was implemented. When ray-cast hits another object, distance to the closest acupuncture node, is calculated. This distance is then, based on the camera distance, evaluated as close enough to be a wanted click, and therefore everything is performed the same way as when directly clicking on the node or is far away and click is discarded.

7.4.4 Showing detail

Displaying detail in AR was a challenging problem. Ideally, the detail should be displayed directly in AR, not to disrupt the experience. It should contain all information from detail displayed as a screen in the application. It also should be facing the user so that the text can be viewed.



(a) : AR with detail of point displayed

(b) : AR options

Figure 7.2: Augmented reality in Self-acupressure application

The following code shows an implementation of a look at constraint that secures that the text is facing the user.

```
let constraint = SCNLookAtConstraint(target: arSceneView.pointOfView)
constraint.localFront.z = 1
constraint.isGimbalLockEnabled = true
node.constraints = [constraint]
node.scale = SCNVector3(scale, scale, scale)
node.moveTowards(node: arSceneView.pointOfView, ammount: Double(5))
node.moveDown(ammount: 1)
```

Listing 7.1: Look at constraint

Displaying of detail was tried in three ways to discover the best solution based on the hardness of implementation and user experience with the given solution. The advantages and disadvantages of these solutions are described in the following sections. All of these solutions would display content on a plane created in the scene kit.

■ View controller

This solution displays a view controller on a plane. The view controller allows further user interaction, like scrolling or navigation. This solution has a problem with incorrectly used threads. It is possible to display the view controller, but undefined behavior sometimes happens as it is updated by the scene view, which runs on background threads, and the view controller needs to be updated from the main thread.

■ AR object

Using an AR object would display text and background as Scene kit nodes. Such a thing is possible, but it is hard as many things have to be handled. The displayed nodes with text are meant for static text or simple dynamic as a counter. There is no auto-layout for text nodes, which means that several things have to take care of. At first, there has to be text wrapping, so the text would not overflow. Then the height has to be calculated so that the other text could be placed below the first one. This would be possible but offers little space for changes and updates.

■ Image from view

This solution was used to display acupuncture points detail in AR. This solution displays images on the plane added in front of the acupuncture point. Image is created from view, which has to be added to the hierarchy; otherwise, the renderer is missing context. This view is added below displayed AR view. Crated image is scaled and added to the plane node with the same aspect ratio.

```
extension UIView {
    func asImage() -> UIImage {
        let renderer = UIGraphicsImageRenderer(bounds: bounds)
        return renderer.image { rendererContext in
            layer.render(in: rendererContext.cgContext)
        }
    }
}
```

Listing 7.2: Extension to get UIImage from UIView

Chapter 8

User testing

The self-acupressure application was tested on six users. The testing was done in two iterations, as the findings from the first three tests found some major issues. These issues were fixed for the second iteration. Table 8.1 shows a list of testers, which iteration they were part of, and the phone they were using. Furthermore, these people were divided into three groups by their skill level with iPhone's. Least skilled is group *Beginner* is meant for people who are using the phone only for basic tasks as calling and using a limited number of applications. The second group was labeled as *Intermediate*. This group is for regular users that know the system and use it on a daily basis for various tasks. The most skilled group was labeled as *Skilled*. This group is meant for iPhone enthusiastic, programmers, and designers. When choosing testers, high diversity was kept in mind in terms of age, skill, and iPhone that they possess.

Tester code	Age	Skill	Phone	Iteration
TesterK12	23	Skilled	iPhone 8	1,2
TesterF1	51	Intermediate	iPhone X	1
TesterZ1	24	Beginner	iPhone SE	1
TesterN2	35	Intermediate	iPhone 8	2
TesterD2	18	Skilled	iPad Pro (3. generation)	2
TesterK2	28	Beginner	iPhone XR	2

Table 8.1: Self-acupressure testers

Tests were done with one user at a time. No information was given to the tester before the start of the test. Summary of tasks is visible in figure 8.1. These tasks were given out by the test conductor. When the task was completed, another task was given out. This was without further intervention of the test conductor. There was also a short interview about application upon completing all of the tasks. This interview consisted of several open questions what would the tester liked to be improved, removed, or added.

1. You have installed a new acupuncture application. Open it and find information about any acupuncture point which you might find interesting.
2. Let's say you have a problem with nose bleeding. Find acupuncture point which would help you with that problem.
3. Find information about where is the point localized.
4. You started to like some points, add at least 3 of them to the favorite screen.
5. You are curious about which points are localized on the head. Display a list that contains all of them.
6. You started to feel, that you liked too many points. Remove the points from the favorite list until there is only one left.
7. Find a tutorial about augmented reality and read it.
 - a. Open augmented reality and display a hand in it.
 - b. Switch option how the hand is showed in the AR.
 - c. Display detail information about some point while still in the AR.
8. Navigate to the point detail, which is on hand.
 - a. You are not sure about the point location, view that point in the augmented reality.
 - b. The hand is too small try to scale it.
 - c. Try to rotate the hand.
9. You found some nasty bug. Try to send a bug report to the developer directly from the application.

Figure 8.1: Task for the testers

8.1 First iteration

Task 1-6 and 9 from 8.1 were, in most cases, done fast without any problems, with the small exception that the tester *TesterF1* has issues to distinguish between the list of problems and list with locations. Some of the tasks were done in other than it was initially anticipated, but that's not a problem as the user can choose his own method to complete the task. More problematic was tasks 7 and 8, which were testing augmented reality. All tester expected that the acupuncture points would be displayed on their hand. They were also navigation through the application very quickly, which meant they could do tasks 1-6 quickly with few errors, but for the AR task, they were little lost when they did not find what they were expecting.

These findings resulted in splitting testing into two iterations, so the problems with AR could be resolved. Improvement done between first and second iteration are in the following table.

- Point detail in AR is opened automatically.
- Improved text in AR tutorial with images.
- Descriptive text added when scanning images.

All these fixes were done to make AR less prone to mistakes and in a way that the user can find a description on how to use it fast.

■ 8.2 Second iteration

The second iteration was done on four testers. Task 1-6 and 9 were done with no problems whatsoever. Sadly the problem with AR persisted. Testers labeled as beginners were slower, but they also carefully read everything on the screen before proceeding, and so they were making fewer mistakes when navigating in application. *TesterK2*, which belongs to that group, had no problems with AR as he read the descriptions carefully, the only issue which he had was being stuck on was rotating the model in AR as he did not know how to perform pinch gesture. The more skilled users were fast and performed most of the tasks in seconds, but they tended to ignore on-screen information. That was not a problem for tasks without AR, where the flow is straightforward. They still tend to get stuck on AR as they were not directly guided on what to do. Furthermore, more skilled tester tends to scan two same images which result in only one side of the hand is showing.

The user feedback after completing tasks was positive. There was some alert about spelling in text, and ideas about UI improvement like better visibility of like button on acupressure point detail screen. They also liked the possibility of the dark mode.

■ 8.3 Future work and improvements

All of these improvements are derived from user feedback and behavior during testing. Most of these improvements are proposed to make augmented reality more apparent to the users. This clarity could be brought with video tutorials or simple animations that would exactly show the user what to do. Another possibility of how to improve the AR experience would be to implement step by step game-like tutorial. Another idea is to give feedback for actions even to disabled elements. For example, at the screen, when the user chooses the bottom of the scanned image to proceed button could do *shake* animation when disabled and elements which are needed to proceed could pulsate to draw user attention. Following list quickly summarizes possible improvements

- Enlarge detail of acupressure point in AR.
- Make AR flow with clear user actions.
- Disabled elements animation.
- Video for as AR tutorial.
- Check if scanned images are different.
- Detail not covered by AR objects.



Chapter 9

Conclusion

This thesis should give the reader enough information about acupressure and existing applications that aim to teach it. With this knowledge, a whole new application was designed and implemented. This design is coherent with iOS guidelines. To see full design go to appendix B.

The application was written in a modified VIPER architecture and should be easily extendable. To increase the speed of development, basic boiler-plate code for VIPER structure was automatically generated. For user convenience, a dark mode was implemented for the whole application. Self-acupressure application is currently available in Czech and English languages.

The augmented reality feature was added to the application to simplify the localization of the acupressure points. This feature uses native ARKit from Apple. The implemented solution is able to display a hand with acupressure points on any detected surface. This hand can also be displayed on scanned images. Other possibilities of how to use AR for acupressure besides implemented solutions are discussed in chapter 7.

The implemented application was tested on five users. Their feedback was taken into account, and changes were made based on this feedback. Conducted user tests can be found in appendix D. Screenshots from the completed application are in appendix C with both light and dark variants.

The final application has been uploaded to the App Store Connect and was reviewed by the Apple review team successfully. The application has not been distributed yet as the acupressure points and their description have not yet been checked by a medical professional. In the future, the application could be extended with more languages and acupressure points. Another extension of the application could be adding more models and points to the augmented reality feature with the possibility of adding animations on how to press the acupressure points.

Appendix A

Bibliography

- [1] Display scene in tabletop ar. <https://developers.arcgis.com/android/latest/java/sample-code/display-scene-in-tabletop-ar/display-scene-in-tabletop-ar.png>. Accessed: 16.5.2020.
- [2] Mvc-basic. <https://commons.wikimedia.org/wiki/File:MVC-basic.svg>. Accessed: 16.5.2020.
- [3] Your guide to the best services. <https://www.reviews.com/tech/consumer/using-dark-mode/>.
- [4] First commercially-available ar navigation app. <https://ssvar.ch/apple-ar-blippar-charts-new-territory-with-launch-of-first-ar-navigation-app-for-iphones-ipads/>, November 2018. Accessed: 16.5.2020.
- [5] What's new in arkit 3? <https://orangeloops.com/2019/09/whats-new-in-arkit-3/>, Nov 2019.
- [6] Acupressure. <https://en.wikipedia.org/wiki/Acupressure>, May 2020.
- [7] Apple unveils arkit 2. <https://www.apple.com/newsroom/2018/06/apple-unveils-arkit-2/>, Feb 2020.
- [8] Rida Aftab. Viper architecture using rxswift. https://medium.com/@rida_36291/viper-architecture-using-rxswift-9a006bc7f8f3, Dec 2018. Accessed: 16.5.2020.
- [9] Fabrizio Brancati. Swift package manager vs cocoapods vs carthage for all platforms. <https://www.codementor.io/blog/swift-package-manager-5f85eqvyg>.
- [10] Edzard Ernst and Myeong Soo Lee. Acupressure: An overview of systematic reviews. *Journal of pain and symptom management*, 40:e3–7, 10 2010.
- [11] Michelle Fitzsimmons. What is arkit 2? here's what you need to know about apple's latest ar update. <https://>

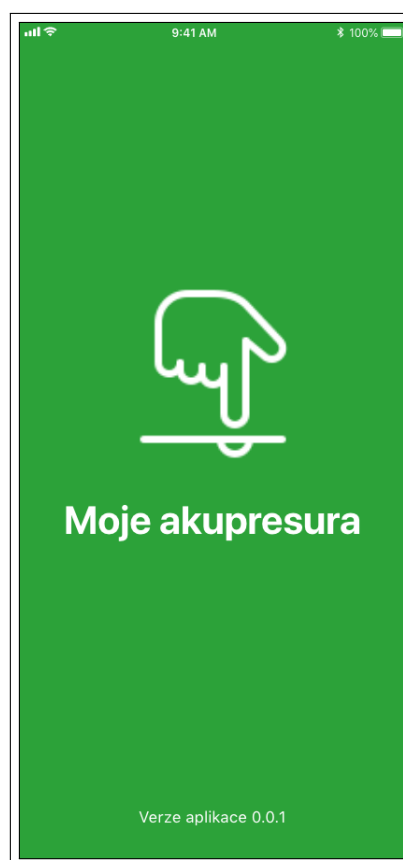
- [//www.techradar.com/news/what-is-arkit-2-heres-what-you-need-to-know-about-apples-latest-ar-update](https://www.techradar.com/news/what-is-arkit-2-heres-what-you-need-to-know-about-apples-latest-ar-update), Jun 2018.
- [12] Apple Inc. <https://developer.apple.com/design/human-interface-guidelines/ios/overview/themes/>.
- [13] Jiří Janča. *Reflexní terapie*. Eminent, Praha, 2. vyd edition, 1996.
- [14] Petr Ješke. Android application for acupressure, 2020.
- [15] Carl Llor and Lars Bjerrum. Antimicrobial resistance: risk associated with antibiotic overuse and initiatives to reduce the problem - carl llor, lars bjerrum, 2014. <https://journals.sagepub.com/doi/full/10.1177/2042098614554919>.
- [16] Amy O'Donnell. Clinical effectiveness: acupressure, acupuncture and nerve stimulation. <https://www.ncbi.nlm.nih.gov/books/NBK390515/>.
- [17] Vojtěch Pajer. Speech development application. <https://dspace.cvut.cz/bitstream/handle/10467/82334/F8-DP-2019-Pajer-Vojtech-thesis.pdf>, 2019.
- [18] Neha Sharma. Mvvm design pattern. <https://medium.com/codewave/mvvm-design-pattern-c5d9f4a10758>, Jul 2019. Accesed: 16.5.2020.
- [19] E Werntoft and A K Dykes. Effect of acupressure on nausea and vomiting during pregnancy. a randomized, placebo-controlled, pilot study. <https://www.ncbi.nlm.nih.gov/pubmed/11584487>, Sep 2001.

Appendix B

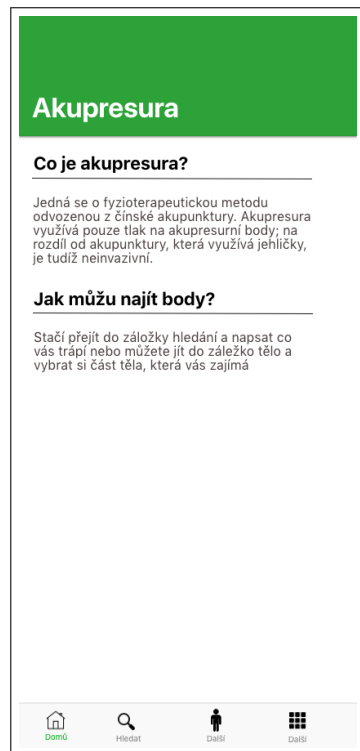
Design



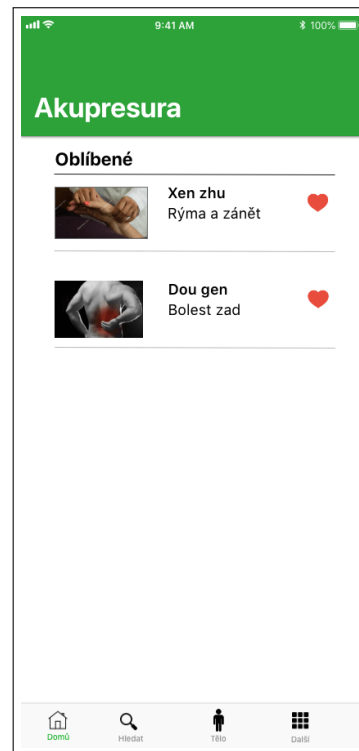
(a) : Part of on-boarding



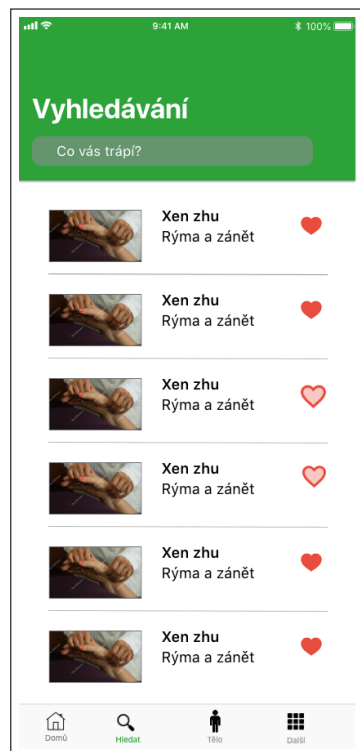
(b) : Splash screen



(a) : Empty landing page



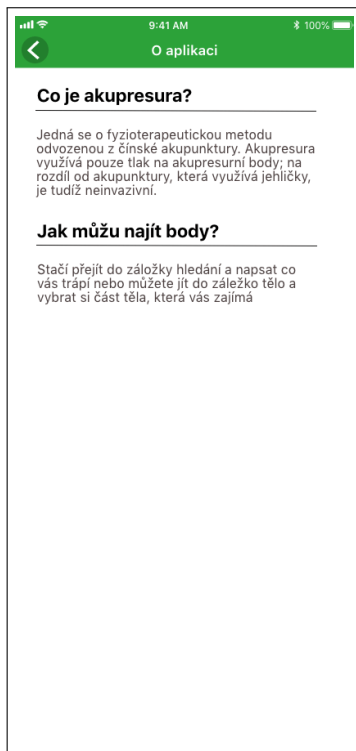
(b) : Lading page with like points



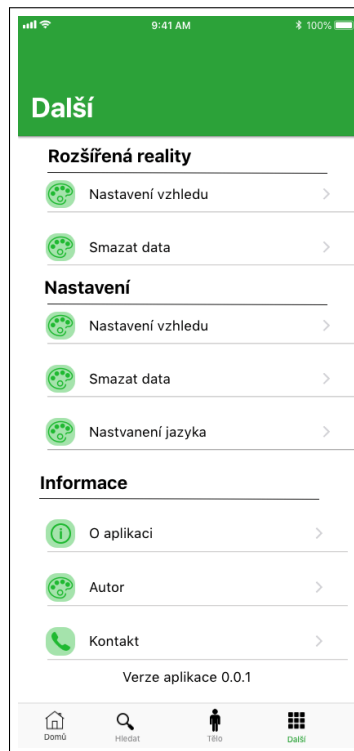
(a) : Search problem screen



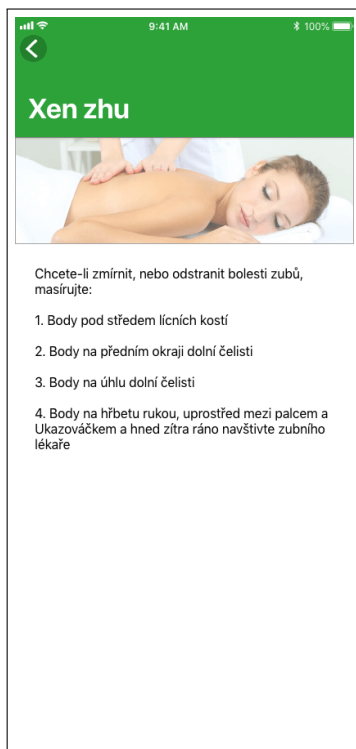
(b) : Choose location screen



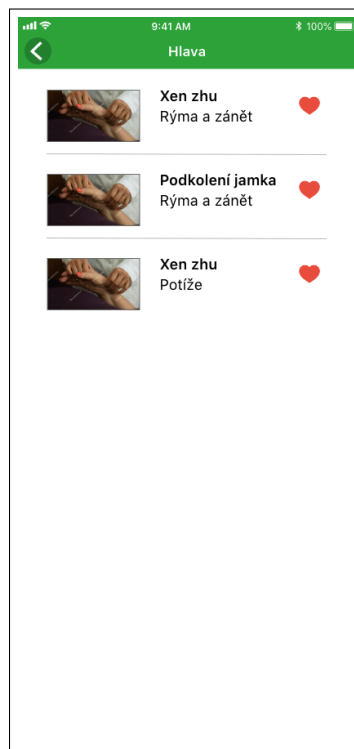
(a) : About screen



(b) : Settings screen



(a) : Detail screen

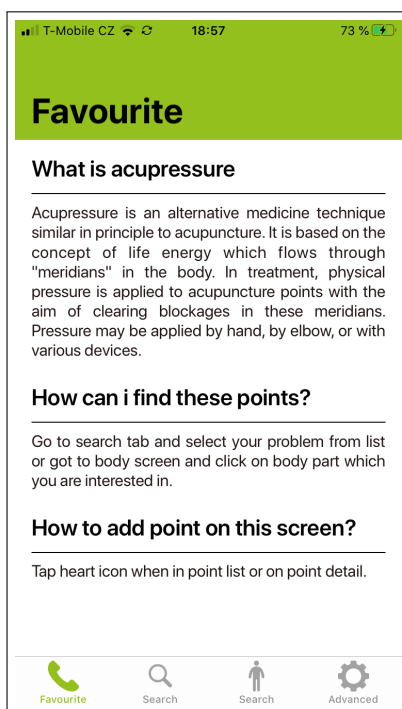


(b) : List of grouped points by location

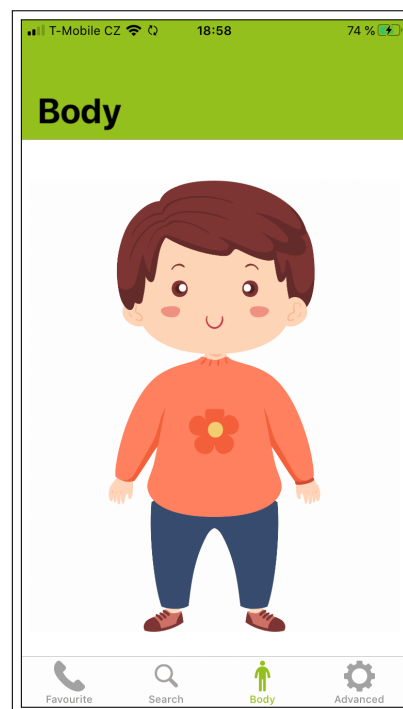
Appendix C

Self acupressure screenshots

Set of screenshots from application taken on iPhone 8 with iOS 13.3 .

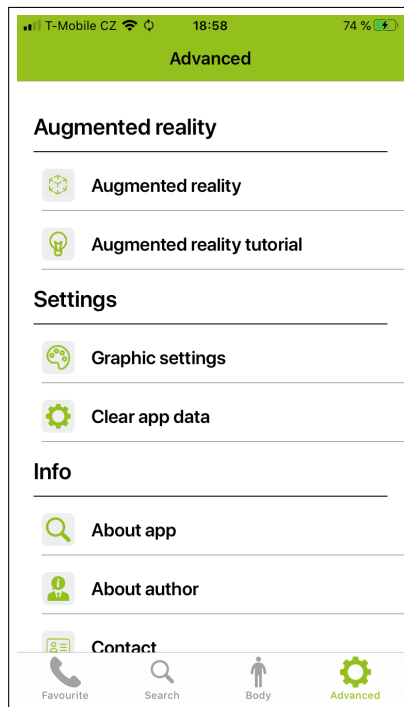


(a) : Default main screen

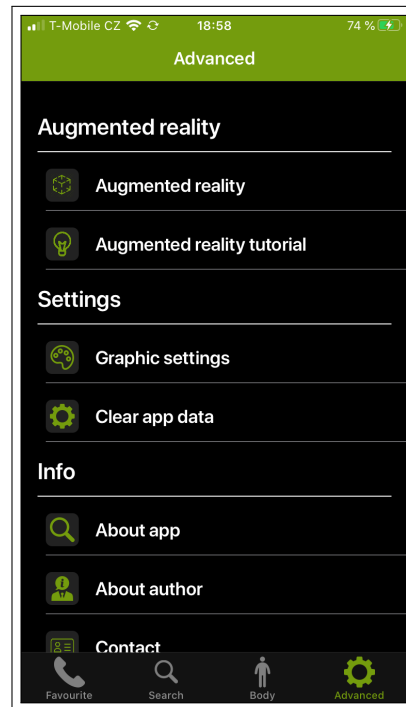


(b) : Default body screen

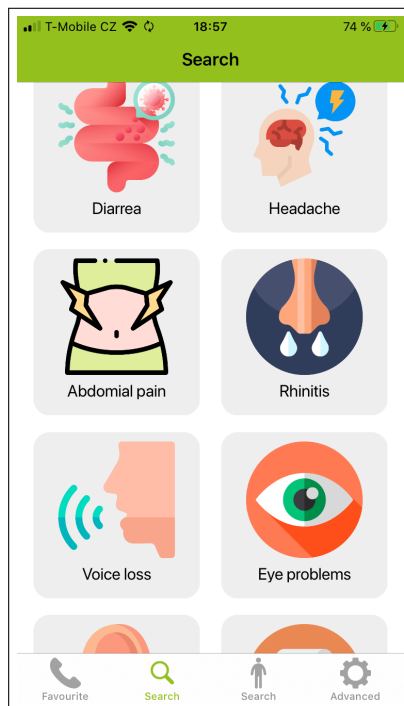
C. Self acupressure screenshots



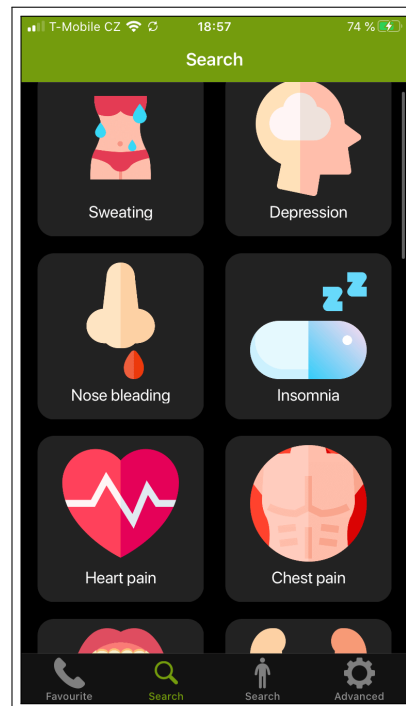
(a) : Default settings screen



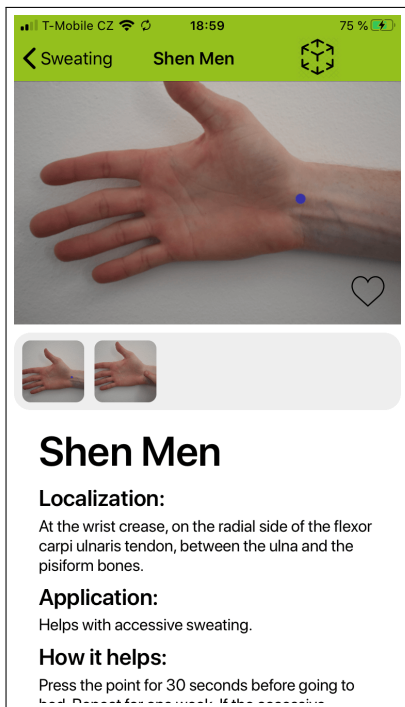
(b) : Settings screen in dark mode



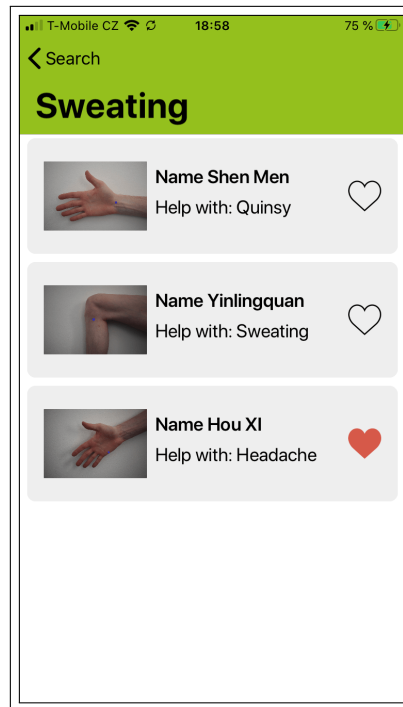
(a) : Default search screen



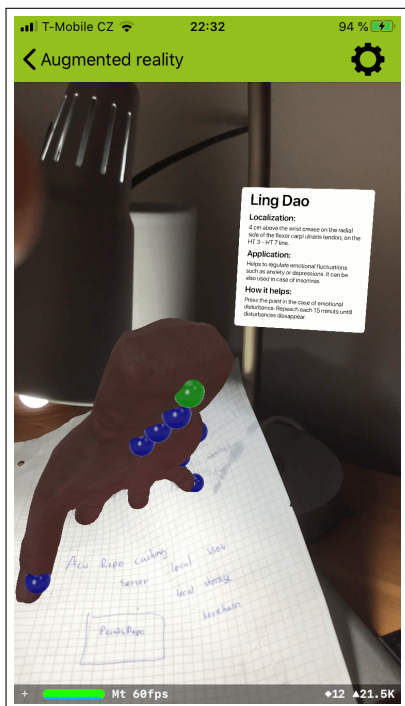
(b) : Search screen in dark mode



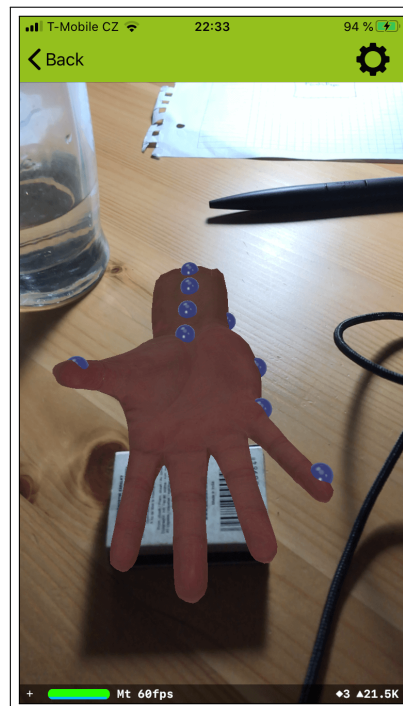
(a) : Problems list screen



(b) : Detail screen



(a) : Augmented reality with detail showed



(b) : AR image detection