ulm university universität
uulm

Universität Ulm | 89069 Ulm | Germany

Faculty of Engineering, Computer Science and Psychology
Databases and Information Systems Department

Enhancing a Cross-platform Application for Internet- and Mobile-based Interventions by Introducing a Communication Module

Bachelor's thesis at Universität Ulm

Submitted by:
Marius-Lukas Ziegenbein
marius-.ziegenbein@uni-ulm.de

Reviewer:
Prof. Dr. Manfred Reichert

Supervisor:
Robin Kraft

2019

Version from November 17, 2019

Composition: PDF-LaTeX 2_ε

Abstract

To deal with the annual growth of patients with mental health problems, *eHealth* systems, and in particular, internet- and mobile based interventions or *IMIs*, were introduced. At the Institute of Psychology and Education, Department of Clinical Psychology and Psychotherapy at Ulm University (KLIPS), IMIs are researched extensively and for this reason, a project was founded to develop a system, which would utilize IMIs. In this bachelor's thesis, we introduced an communication module for this project. The two categories of problem statements regarding this introduction consist of the research and the technical part for this communication module. In the research section, we inspect, how communication enhances IMIs and how we can utilize guided interventions through it, by being able to receive feedback and send messages or questions according to it. The technical section on the other hand involved the process of implementing this module to the project. Consequently our objective was, to introduce said communication module to this project and enable guided interventions as well as communication between the patient and eCoach in the project. This included the design and implementation of this feature and thinking through the process of how this could be done. To do this, we analyzed related work, including other mental health and instant messaging applications, and started to build the structure. Other contributions included the update of Ionic 3 to 4, setting up other smaller features and setting up the base for push notifications. In conclusion, the challenges regarding the implementation were stated and an outlook was given for features and aspects that could be improved upon.

Contents

1

Introduction

A reoccurring problem in Germany and in other countries is, that the waiting times to get a consultation in the mental health sector is quite lengthy. According to a study of the Bundes Psychotherapeuten Kammer (BPtK), the average waiting time in Germany in 2011 was 19,9 weeks, 5 months of waiting for people suffering illnesses that they might not be able to fix themselves [1]. While the waiting time to get into the first consultation decreased, the waiting time for the actual therapy is still tedious. This is the result of the general shortage of psychotherapists in many countries. At the same time, the number of adults that suffer from mental health problems increased significantly over the last years [2, 3]. These disputes together result in a significant deficit of mental health treatments of people around the globe. In Figure 1.1 we can observe, that by the year 2025 the shortage in the whole Mental Health Service sector will persist and intensify. For example, in the sector for psychiatrists, psychologists and mental health counselors the predicted shortage stands at roughly 100.000 in the United States.

In order to fill this gap, the thought to introduce and provide therapy through the internet, applications and our connected devices could be effective. In this world, where we already do many things in our daily life with our smartphone or on desktop like ordering food [5], request transportation [6], make doctors appointments [7] or even replace the whole process of going to the doctor itself [8]. When recognizing this, the thought of also treating mental health problems through this medium is not far fetched. The introduction of such a service could be beneficial to the patient, the doctors and even the health insurances because of its cost and time effectiveness for all participants [9]. Consequently, Internet- and mobile-based interventions, for short *IMIs*, were created.

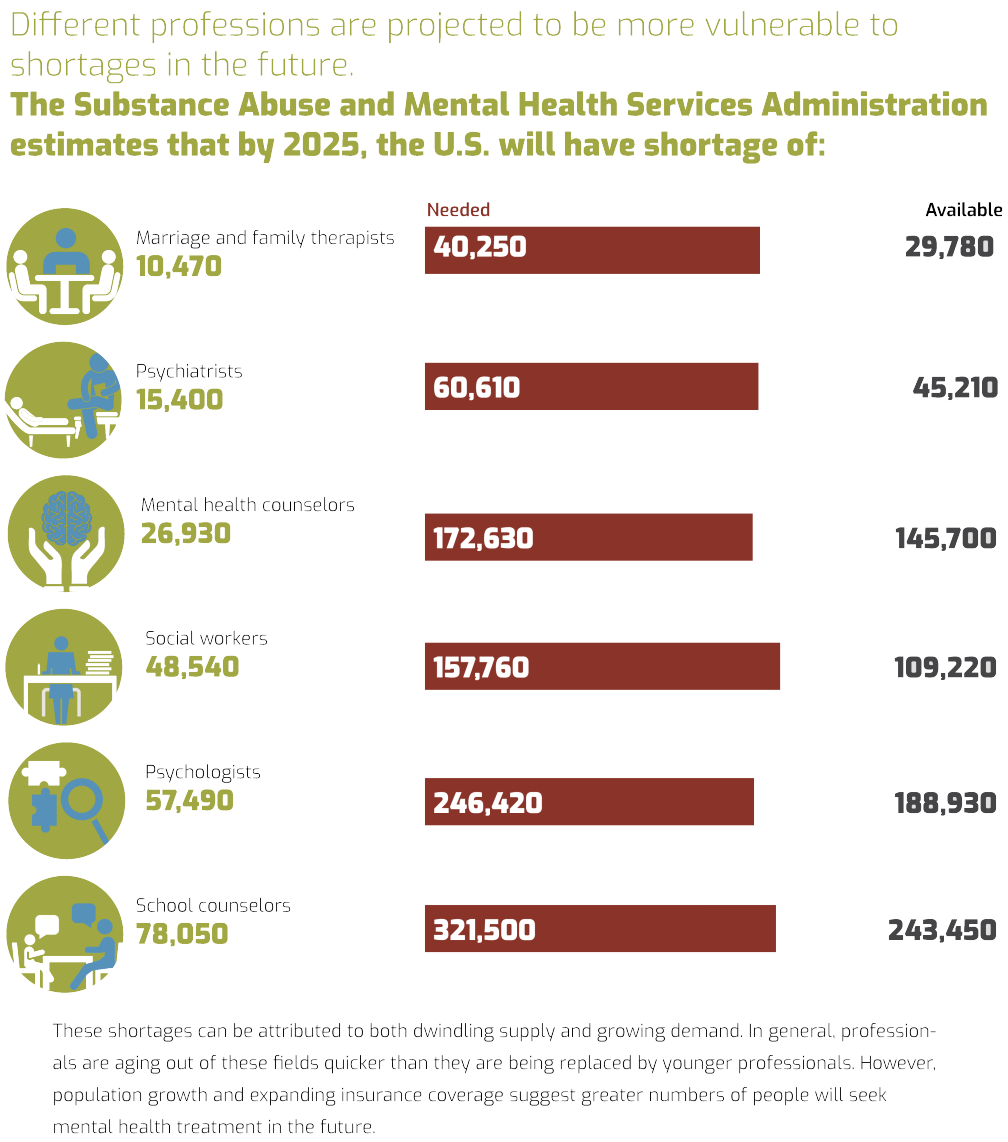

Figure 1.1: The prognosed shortage of therapists in different sectors in the US [4]

Internet- and mobile-based interventions are a relatively new addition in the world of mental health treatment. The effectiveness of treating illnesses like anxiety, insomnia, depression, tinnitus and problematic alcohol consumption with IMIs has been established in many different studies [10, 11, 12, 13]. Furthermore, because of the cost, time and alleviate effectiveness of IMIs, they are being used more and more often. At the Institute of Psychology and Education, Department of Clinical Psychology and Psychotherapy at Ulm University (KLIPS) IMIs are being researched extensively. For this reason, a new project was started, which should build upon the idea of IMIs and allow the therapists, researchers and other involved ones to attain more insights. In this work, we will enhance this project by introducing a communication module. This allows the patient, to receive feedback and start conversations on interventions or the given feedback. Through this feature, we expand the project to utilize guided interventions, which are, in comparison to unguided interventions, much more beneficial for certain mental health disorders. We will provide a more detailed look for this in Chapter 2. Consequently, through this enhancement, another method for treating patients using IMIs can be accessed.

1.1 Parts of the system

This project is divided into four parts, each representing a subset of the whole system. The project therefore includes an application programming interface (API), the backend system of the project, an Content Management System (CMS), in which interventions can be created and edited, the eCoach platform, through which the assigned therapists can look after their patients and finally, the patient app itself, which is the component of the project, which patients would use to take part of these interventions, interact with their eCoaches and work through their interventions.

1.2 Problem Statement and Objective

In this section, we will shortly clarify the problem statement and objective of the work.

Problem Statement With the current iteration of the patient app, no communication module has been implemented. The Problem here lies within the inability to offer guided interventions[1]. Depending on the use case and the illness that needs to be treated, guided interventions can have a much bigger and positive impact on the well being of the patient than with unguided ones. For this reason, in this iteration the app needed to be enhanced to include said module. The communication in guided interventions expresses itself mainly through feedback of the eCoach, but also for a chatlike interaction between these two as well as for patients with the same condition. Now to the technical aspect. The problem and difficulty with this chat module is, that on one hand, the server can not send notifications to specific devices, on the other the app can not listen for changes in the server and generate a Notification based on newly received information.

For this reason, we had to think of a solution, how to receive these messages in a efficient way. The two possibilities that came to mind were the following. The app either has to pull every 10 seconds (This restriction allows the chat to be called instant) new data from the server to receive messages and other information or to implement an external push notification service, that would get data from the server whenever a new message was received and distribute them to the whoever should get that information through push-notifications.

Objective The Objective therefore is to implement a feature, through which the patient can receive new information of all sorts, for example new messages, feedback on a intervention or new interventions, sent by the server and access them. This allows the possibility to provide guided interventions and therefore have better results in certain therapies.

[1] An explanation of guided interventions will be available in Chapter 2

1.3 Structure of the thesis

We begin by elaborating the fundamentals in Chapter 2, in which we describe eHealth and interventions and their benefits. We talk about the platform that we used used, namely Ionic, which enables the cross platform functionality and go through some related work in the sector. After this we name the requirements in Chapter 3 that specify the features of this iteration and describe them. Additionally we describe the non-functional requirements. Once these features are clear, we classify the design process in Chapter 4. How the app will be structured, which components will be used and which components and pages will be added. Following this, the biggest part: The Implementation in Chapter 5. We approach the added components, show the UI and explain the methods and communication they use to enable them. Furthermore, we will go through the process of upgrading Ionic from version 3 to 4. Consequently, in Chapter 6 we discuss the whole process and its difficulties, explain a bit on future implementation and lastly compare the requested and implemented requirements and discuss their progress. Finally, a conclusion of the whole thesis and a short outlook will be given in Chapter 7.

2

Fundamentals

In this section, we want to elaborate the fundamentals of this thesis. We will talk about internet- and mobile-based mental health interventions, from now on named *IMIs*, how they work, when they are used, how they separate into two categories and therefore the effectiveness of these categories and how our communication module would benefit the application. Thenceforth the backbone of our system, the cross platform framework. A clarification will be given on the practical approach of these frameworks, when they are used and comment on their advantages and disadvantages. Finally we will look into related work, inspect their ideas and means and perchance adopt some of their practices.

2.1 Internet- and mobile-based mental health interventions

The effectiveness of Internet- and mobile-based mental health interventions has been proven in many different studies and research. IMIs are used in the mental health treatment, for example for disorders like anxiety, depression, stress and sleep problems. These and many more mental disorders affect one in four people, however just 43 percent of people with mental illness received treatment in 2016 [14, 15, 16]. Some reasons for this were broached in the introduction like the shortage of therapists and concurrently increasing number of people with mental health issues. Consequently, it is desirable to facilitate more people to recuperate from their ailments. IMIs try to solve this problem. IMIs work on the premise to introduce mental health treatment through electronic devices, hence *eHealth*.

2.1.1 Why and how IMIs could help

There are different aspects and reasons, why IMIs could help with this problem. One reason, why IMIs could help to improve the utilization is by their anonymity. Furthermore, the need to go into the doctors office, or make a phone call, recedes, since IMIs can be accessed to at their own convenience. E.g., if a patient finds himself in a situation, that could potentially trigger their anxiety, like an social phobia patient going to a big event, the patient could access the treatment, calm down and loosen up. Adding to this point, the patient can set their own pace in privacy [17].

2.1.2 Interventions

For a better understanding of IMIs, we will now explain, what interventions itself are. Interventions, in principle, are defined as any sort of psycho therapeutic treatment, mainly through talking therapy, with the intention to improve outcomes. In the progress of intervention treatment, the goal is to change the behavior, provide knowledge of the medical condition and improve upon it. It is the process to bring change, alleviate the symptoms and target the root of the disorder [18]. In eHealth, interventions mostly act as the combination of filling out studies and being able to talk about them with an assigned therapist. In guided interventions this interaction extends through feedback, that the therapist can give on the answers of the study. Furthermore, in many cases a journal of some sorts and or a mood board is apart of this program. Since the difference of guided to unguided interventions can be considerable, we will now discuss them and explain the differences.

Guided interventions

In Internet- and mobile-based mental health interventions a common structure would be, that there is an Content manager, the author of interventions, an *eCoach*, the therapist of patients and patients themselves. In this structure. a guided intervention would be, if a patient gets an eCoach assigned, who would assign interventions to the patient. If the

patients fills out the assigned interventions, the eCoach would have a chance to review his input and give feedback according to it. Furthermore, the eCoach may unlock more or different interventions based on the patients input.

Unguided interventions

Unguided interventions on the other hand are interventions, in which the patient gets no human feedback or support. There may be instances where the patient gets automated responses based on his input by bots, but for now that will not be relevant. This means, that unguided interventions represent self help.

Difference between guided and unguided interventions

Since the difference between the guided and unguided interventions is human input from a therapist, there are different use cases for these types. In study [12] and [19], you can see, that for different sicknesses, disorders or social phobias, one or the other kind of intervention is more beneficial to the effectiveness, cost or satisfaction. E.g,. guided may be better suited for social phobias or depression, while it makes little to no difference when treating problematic alcohol consumption. For this reason, the possibility to offer both types of interventions is advisable.

2.2 Cross platform framework

In order to enable different users and their preferred *platform*, smartphone or desktop and in some cases both, there needed to be a solution to minimize the needed resources. For this reason, the app was designed under the concept of cross platform development. Cross platform applications are designed to work on multiple systems. They work by compiling the implemented code into that of the different platforms. On Web it generates a normal web page, with routing, all the resources and responsive content. On Android it generates an *apk*, which can be installed natively or by an emulator, in Ionics case

cordova. And on *iOS* and through Xcode, the native coding platform for mac, it can be built into an iOS app.

The advantage of this cross platform development is, that you can code and implement an application in such a framework, which translates it for these different systems, thus effectively removing two platforms that you would normally have to implement additionally. This not only saves time, but also resources when developing and furthermore, when maintaining the project.

The disadvantage on the other hand is, that the cross platform development can be much more difficult than native ones. Through this framework, some things work differently than we would be used to. You have to keep in mind, that you are virtually working on three different terminals, and not on all of these terminals can use the same assets. For example, a normal push notification, that would just get send into the tray of and android or iOS device, would have to be handled differently on desktop. And the UI and UX has to be thought through so that it looks and feels good on a small device like a smartphone but, also scales up for the desktop.

2.3 Related Work

One of the existing applications that works with IMIs is minddistrict [1]. This system is currently in use at the Institute of Psychology and Education, Department of Clinical Psychology and Psychotherapy at Ulm University (KLIPS). Similarily to the project we develop, minddistrict is an eHealth platform, through which interventions can be developed and eCoaches treat their patient. In the Figures 2.1, 2.2 and 2.3 we can see, how minddistrict looks like. Starting with Figure 2.1 we see the catalog of mindisstrict, in which the interventions are stored and where the patient can access them. The patient has the opportunity to look at them and write about them. In Figure 2.2 the continuation point is visible. On this page, a short summary is available and there may be praise on the current progress. This as well is the point, on which the patient can start a conversation and ask questions about the intervention. In this menu, the patient

[1] https://www.minddistrict.com/de-de

can navigation between the homepage, the catalog, the conversations and their profile. Lastly, in Figure 2.3, the chat is observable. We notice, that the structure is similar to other instant messaging applications. In the Chapter 4, we will come back to this, when designing our own chat.

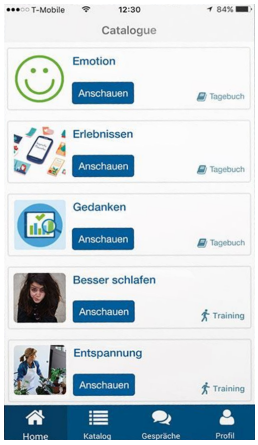

Figure 2.1: The catalogue in the mind-district app

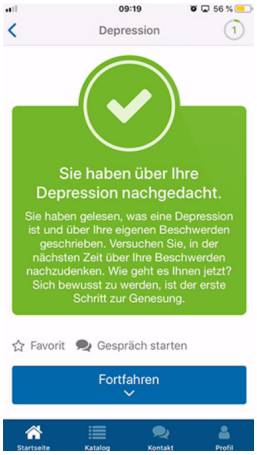

Figure 2.2: One of the lections

Figure 2.3: The chat module of minddistrict

Another application is Mental Health Interventions [2]. This app acts more like an addition to a therapy, through which the therapist can track the well being of the patient by getting insights of mood, journals and through voice recordings. Since this app is rather barebone, we will move to the next one.

Youper [3] is a mental health app, which more closely resembles the unguided interventions and self help. However you have an assistant, with whom you can communicate, track your mood and track for example your anxiety levels throughout the day. In this app, there is also a journal, in which the patient can write situations of the day, how it got handled, possible problems and challenges and more. Through all these things you also have a history which can show the effect of this therapy. The Figure 2.4 represents some of its UI. We can see, that the chat of this application is rather rounded and simple to read. The other pages show their interpretation of a moodboard and one of the methods to calm down the user.

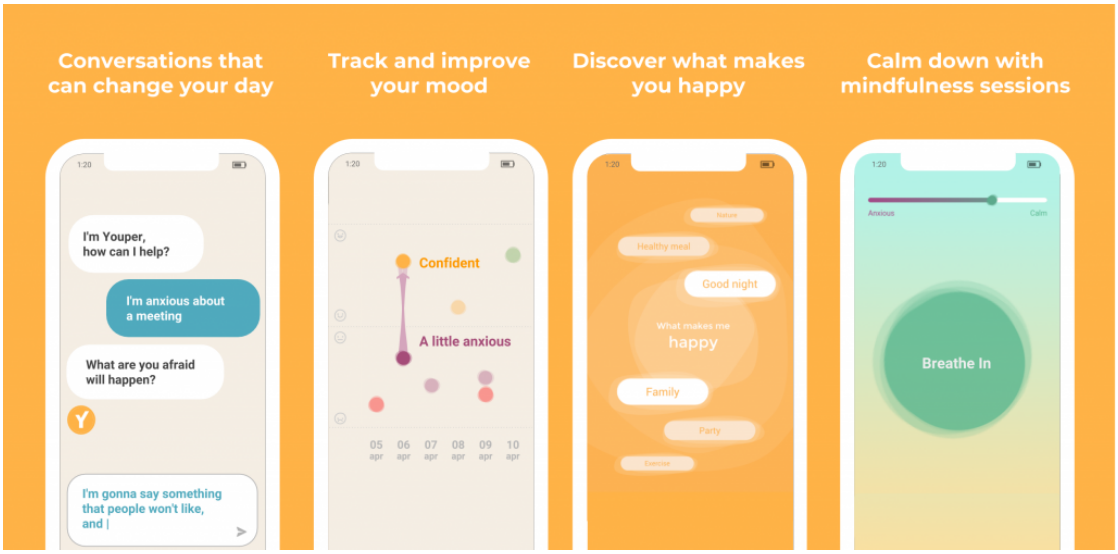

Figure 2.4: Youper and its components

[2] https://mentalhealthintervention.org/
[3] https://www.youper.ai/

3

Requirements

Requirements are a description of features and functionalities in a system. They describe these features, what they do, what the reason for them is and how important they are. For the prioritization of our requirements, we will use the MoSCoW method [20]. We can use this method to specify the importance of our requirements. It orders them in the following way: M means Must have, so if a requirements has this type of prioritization is has to be in the final product and is critical for this iteration. S stands for Should have, thus requirements with this tag are quite important and are desirable in the iteration of the product. The C symbolizes Could have. Basically, if a requirement with this label is implemented, it's a nice addition if the time and resource constraints allow for it. Lastly, W represents Won't have. Requirements that get this tag are not supposed to be included, either because it's not critical and will be implemented in a later iteration or because it was decided against.

3.1 Functional Requirements

ID:	1
Title:	Chat
Description:	A chat that allows the patient to receive and send messages in the app. Most importantly between the patient and eCoach, but also for communication inside groups of patients with the same intervention type. The messages should be instant (Dealed with within 10 seconds)
Reason:	This enables the communication between patient and eCoach and other patients inside the app.
Prioritization:	M

ID:	2
Title:	Basic Feedback Function
Description:	This basic feedback function should enable the patient to look at a feedback sent by the eCoach. This basic feedback would consist of text, on which the patient can answer and, if needed, look up the intervention or answersheet, to look at his inputs, may he have forgotten. This feedback would be presented, whenever the patient send a completed answersheet of an intervention and the eCoach answers to this answersheet.
Reason:	This enables the possibility of an guided/accompanied intervention.
Prioritization:	M

ID:	3
Title:	Enhanced Feedback Function
Description:	The enhanced feedback function will then improve the basic one. At this point, this function will then be used, whenever the eCoach set a breakpoint on a intervention, in which the intervention would pause for the patient. When paused, the patient gets an information popup, which states that the intervention is paused and must be reviewed by an eCoach. The review consists of inspecting the patient inputs and leaving the feedback. Once the eCoach reviewed the input and sends feedback, the patient would get a notification. After this process the patient can continue the Intervention.
Reason:	This enables the possibility of an enhanced guided intervention.
Prioritization:	W

ID:	4
Title:	Notification Service
Description:	The App should be able to send Notifications to the user, whenever new Information got sent from the server. Notifications would include: -Feedback by the eCoach on an intervention of the patient -New lesson available -Weekly intervention -Reappearing interventions -...
Reason:	This allows the application to inform the patient of new information.
Prioritization:	M

ID:	5
Title:	Push-Notifications
Description:	The App should be able to transform Push Notifications from the Server and generate notifications. This means, that whenever the patient is not currently using the application, the patient should get a push-notification to his device and therefore be informed, whenever he got new information.
Reason:	This allows the application to remind the patient of Information even if he isn't actively using the app.
Prioritization:	M

ID:	6
Title:	Remember me function
Description:	The patient can set a reminder on a chosen intervention. Once a reminder is set, a notification will be delivered in the given timespan or at a given time. This notification could be set, to remind the patient to continue his intervention when left off or when he wants to do the intervention at a later time.
Reason:	For a reminder of important or wanted interventions
Prioritization:	C

ID:	7
Title:	Watch list for interventions
Description:	The app should have a watch list in which the patient can save interventions that he may want to do in the future. Furthermore, this list could show a list of interventions, that the patient should do later on in his therapy assigned by the eCoach.
Reason:	Ease of use for the patient so that he doesn't have to search liked interventions repeatedly.
Prioritization:	C

ID:	8
Title:	List of all interventions
Description:	A list of all interventions can be shown in the app, so that the patient can look up which interventions exist.
Reason:	Transparency and display of potentially fitting interventions.
Prioritization:	S

ID:	9
Title:	List of open interventions
Description:	This list should show all open interventions in which the patient can sign up by himself.
Reason:	Ease of use for patient. The patient can decide, which interventions he wants to take when following an unguided therapy or in addition to the guided one.
Prioritization:	S

ID:	10
Title:	Filter function for interventions
Description:	In the list of all interventions/open interventions, the user should be able to filter this list with given keywords like Depression, Tinnitus, Eating Disorder etc.
Reason:	Ease of use for patient, patient doesn't have to go through all interventions to find fitting ones.
Prioritization:	W

ID:	11
Title:	Basics of conditional prompts
Description:	The app has to handle simple conditional questions in the interventions. Simple conditional questions and their processing would be True and False prompts, check if one input is equal to the given case of the author and generate the corresponding new prompts. Example: Yes/No question, if answer is Yes then generate more fields which further ask the background, if No then continue.
Reason:	For a basic representation conditional prompts in the app
Prioritization:	S

ID:	12
Title:	Improving upon the conditional prompts
Description:	The app should be able to handle more complex conditionals existing in the interventions.
Reason:	To enable the correct representation of some interventions.
Prioritization:	W

3.2 Non-Functional Requirements

Title:	Accessibility
Description:	The system shall be accessible to people with access to both the web and smartphones.

Title:	Availability
Description:	The system shall be available to use in 95 out of a 100 cases. If the system is unavailable, the user should be informed.

Title:	Usability
Description:	The system shall be designed in a way, through which patients of all sorts can understand the features of the system. This should be achieved, by using common UI norms and standards.

Title:	Scalability
Description:	The system shall be designed in a way and with suitable components, through which a high scalability can be achieved. High scalability meaning, the the system shall still work, even if 500 patients use it at the same time and enable all of them to to use the application without major problems.

4

Design

In this Chapter, we will explain the design of the features, what pages and components will be added, what the UI will look like and how each of these components will be structured.

4.1 Structure

We start by adding the **threads**, **chat overview**, **feedback overview**, **chat** pages and the optional ones for **open and all interventions**. Furthermore we will add the *providers* for the messages and threads into our system and improve the notification service to fit our needs. The Figure 4.1 will represent the general architecture of the system and how the components will interact. While the faint components represent the existing structure, the clear ones stand for the added ones. A further explanation will follow, but first, we will look at related work in the means of our chat and how it could be designed.

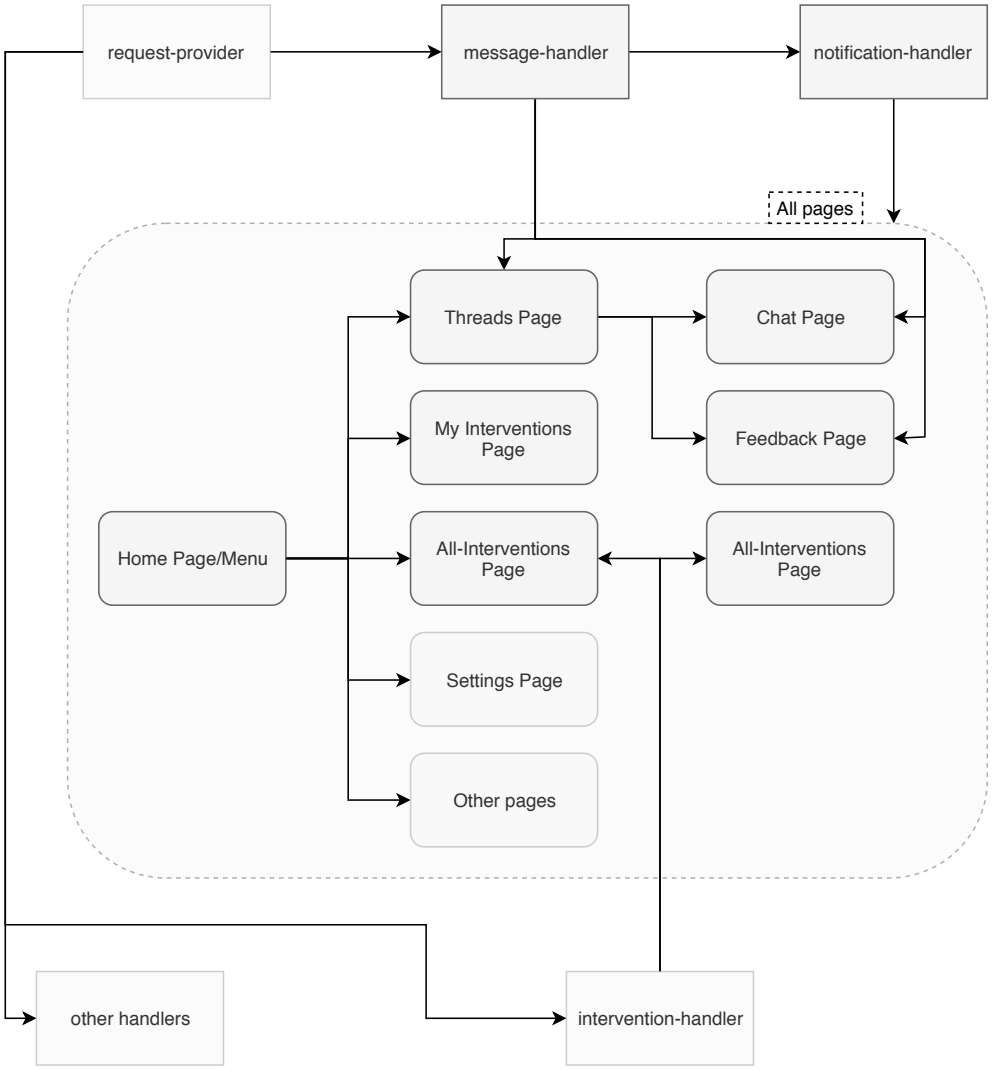

Figure 4.1: The architecture of the app with added communication module

4.2 UI of related work

One of our main goals is to implement a chat, in which *eCoach* and *patient*, but also other patient in general, can communicate. For this, we would like to know how other services designed their chat service, focusing on the UI and UX core components. For this, we will look at WhatsApp [21] and Telegram [22]. Lets start with WhatsApp.

4.2.1 Whatsapp

WhatsApp is an instant messaging service, that is used mostly on mobile, but can also be accessed on desktop through its WhatsApp Web feature.

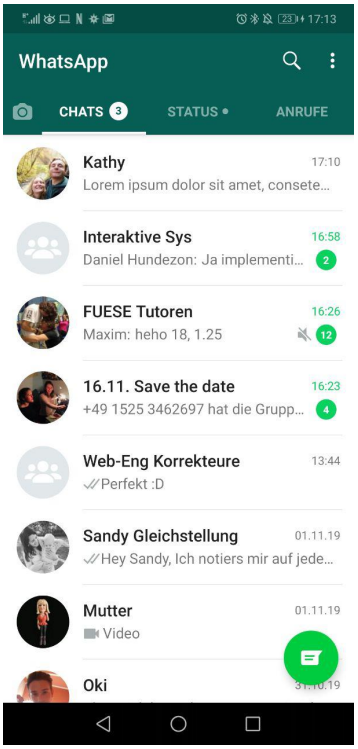

Figure 4.2: WhatsApp on mobile - The Overview

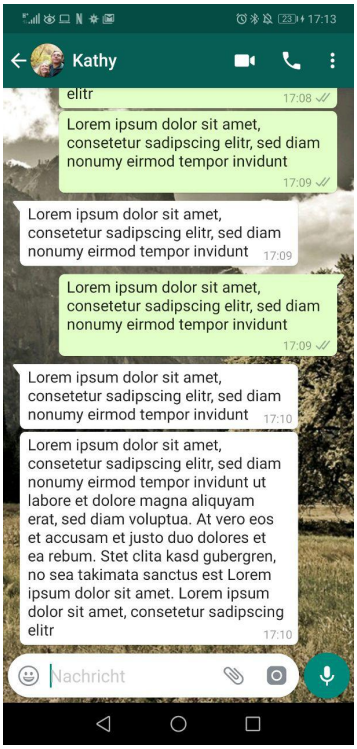

Figure 4.3: WhatsApp on mobile - The Chat

As you can see, on mobile 4.2.1 we have a overview page, which presents all current chats. When clicking on one of these items you can access the chat itself. In here, the

messages get presented as speech bubbles and, based on who the sender was, these bubbles will be located on either the left, if it was sent by someone else and on the right if not. At the same time, the color of the bubble will be different. This makes the UI really easy to read and its clear, who sent which message. As on almost all applications, the header contains a back button, in this case also information about the current chat and some more features. On the bottom, there is the input field for the messages. If it is empty, there will be buttons for a speech message or for taking a picture and general attachments, and if not, we can see the send button.

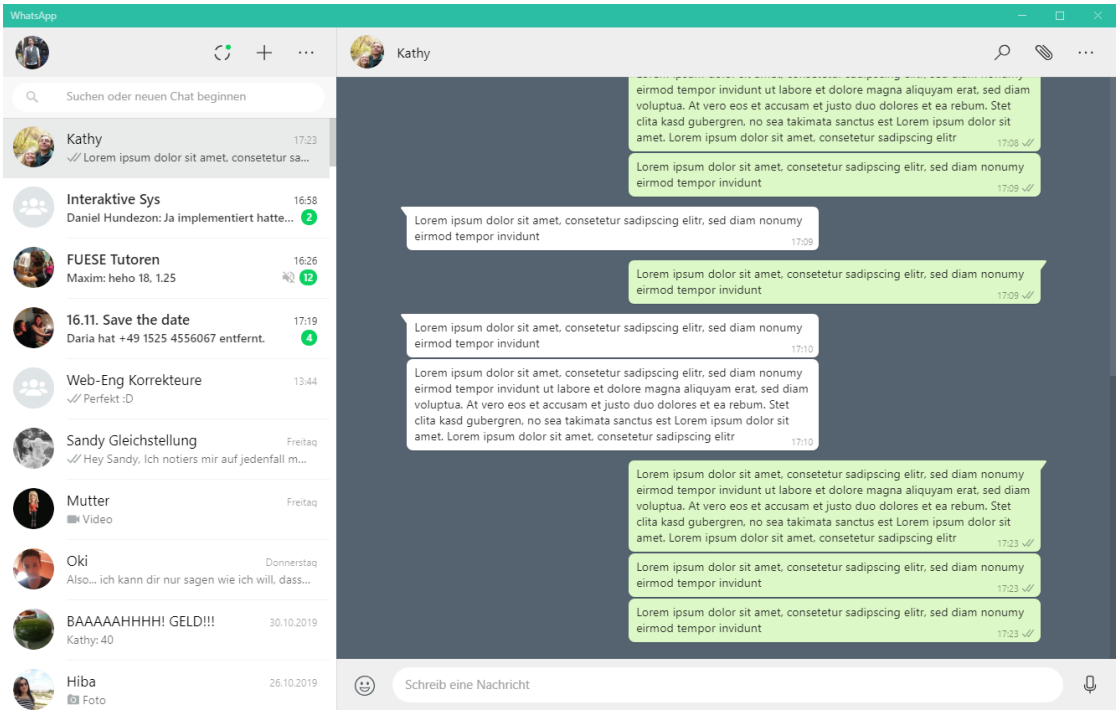

Figure 4.4: Whatsapp on desktop

On WhatsApp web 4.4, things are a little bit different. What first strikes the eye, is the combination of the separate pages overview and chat. On WhatsApp web, the list of chats will not be located on a different page, but instead on the left where it can be easily accessed. Then, when clicking on one of these chats, that chat will open on the right side. The main idea of messages as speech bubbles still apply, however the scaling is somewhat different.

4.2.2 Telegram

Telegram also is an Instant Messaging service. Telegram has its users spread more broadly between mobile and desktop.

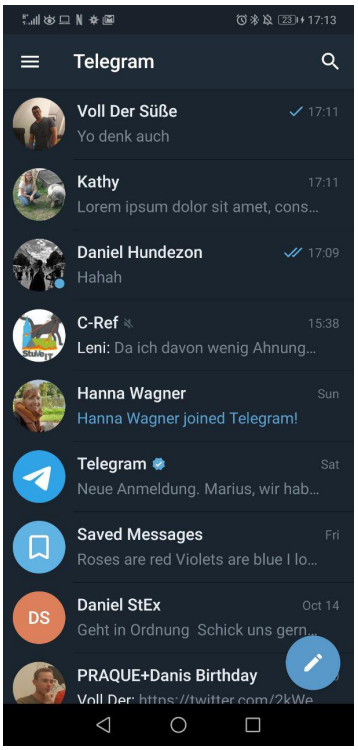

Figure 4.5: Telegram on mobile

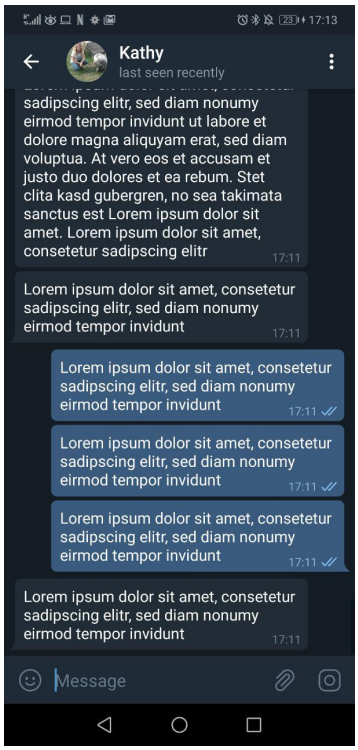

Figure 4.6: Telegram on mobile

We can see, that many UI concepts that were seen in WhatsApp also apply to Telegram. On mobile 4.2.2 the overview page and the chat with its bubbles, header with back button and information and the footer with the input field and some features. Similarly on the desktop app 4.7. We have the overview on the left and then the chat on the right.

Because the main structure on most messaging apps and parts in other app look and feel the same, it would be advisable to use these designs in order to allow the user to instinctively know how to use the system.

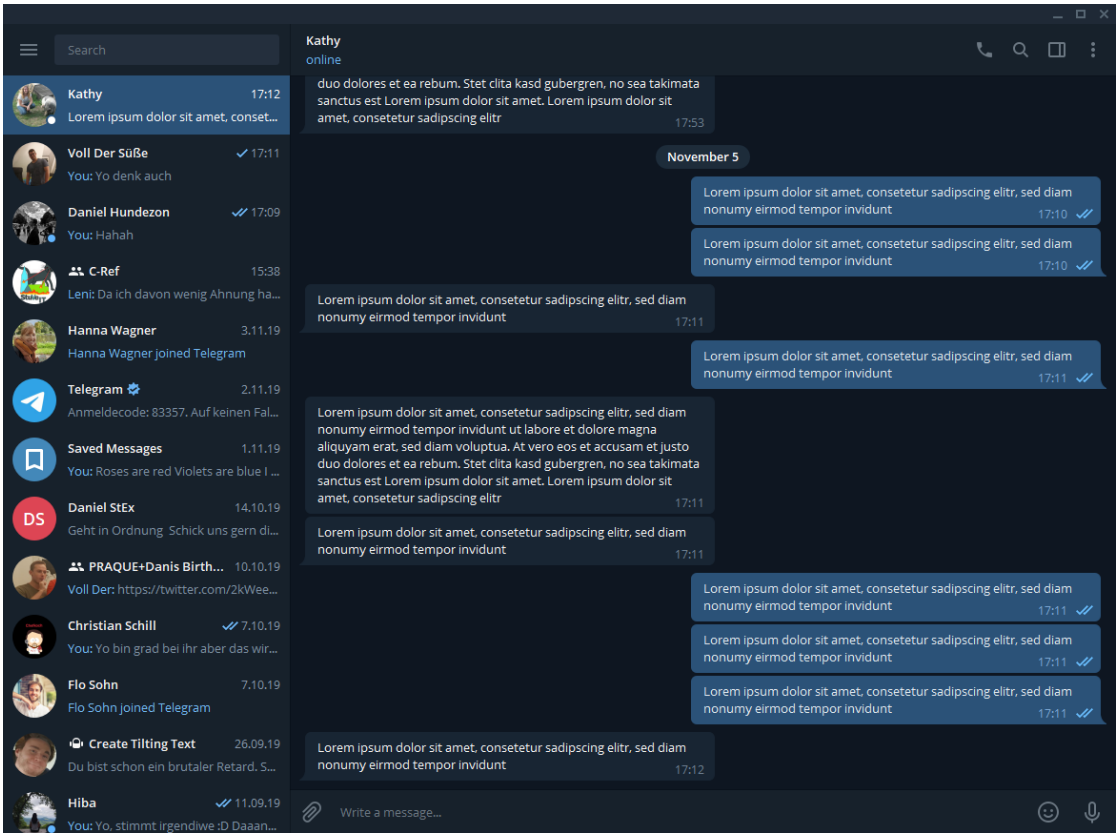

Figure 4.7: Telegram on desktop

4.3 Notification

In order to receive messages there were two possibilities: Either to pull every 10 seconds (In order to abide by the instant messaging requirement), or to integrate a notification service, which would be able to receive messages by the server and distribute notifications to the devices that should receive them. While the first option is less efficient in power and network questions and it would be ideal to implement a notification service, in this iteration we will implement the former option and later append the external notification feature. There are different types of notifications in our system. For the patient app, they could be divided into three parts: Notifications for new messages and feedback, notifications for new interventions, meta information like "You were put into group X" and "X is now your eCoach". For the reasons explained above, we will focus on the notification of incoming messages and, when later when the notifications service is implemented and the server supports it, expand to the other types. An aspect, that we have to keep in mind is, that we have to separate the notifications, that we would get when actively using the application in comparison to having it in the background. Since the user would not want to get a message to the tray whenever he is using the system, we would like to have a system, that handles messages if the system is in use. For this, we will add a *popup* generated by the native *alert controller* and show relevant information on this popup and add a *routing*, through which the user can directly move to the specific page, in our case the **chat or feedback overview** page. In contrast to the message notifications, that can currently only be presented if the app is active, notifications that remind the patient of certain interventions, can be executed differently. For these, we can use the *local notifications*. This means, that whenever the patient sets himself a reminder or if a interventions has to be done weekly, we can call this local notification service and deliver it at a given time. These will then be added to the tray.

4.4 Messages

Before continuing with our components, we want to explain, how the messages work in our system, how they are saved on server side and how we can access them. Starting

of, we use *HTTP requests* to send messages and get them by receiving *thread lists*, which kind of act as groups. In these groups, all messages are stored. If we want to send a message, the user would be inside one of these groups and when sending the message, we will use the *thread ID* of the current group and send a HTTP request with the content of the typed message. Because we are authenticated to the server, we got a *token* assigned, and through sending the message with our token attached, the server can identify us as the sender. At this point, the message gets added to the messages array in the thread.

4.5 Components

Since the original app was not set up to handle messages, we needed to add a few components, that can handle this message traffic. For this we added an *message handler* and improved upon the *notification handler*.

4.5.1 Message Handler

In this handler, most of the message logic is located. Through this service, we load our *threads*, split them into the categories of normal message and *feedback* based on the *answersheet id*, load messages into these threads and implement the sender function. Because of the constant connection to the server to enabling up to date messages, this handler will have a heavy connection to the *HTTP handler*.

4.5.2 Notification Handler

The *notification handler* had to be improved, to support our features. This service will be used, whenever the user decides to be reminded on a later point for a given *intervention*, whenever the user did not finish his intervention or if a new message is available from the server.

4.6 Pages

Furthermore we need to add a few pages, which will show the UI of the newly implemented features. We need pages to enable the chat, consisting of a **chat overview** and the **chat** itself. We need the feedback feature, once again consisting of **feedback overview** and **feedback** itself. And to combine these two into one screen, we added an **thread** page, which will show on one tab the **chat overview** and on the other the **feedback overview**. For the optional requirements **all interventions** and **open interventions**, we also needed a new site to show according information.

4.6.1 Threads

This Page is more of a placeholder for the *subpages* **chat overview** and **feedback overview**. In this page, we will integrate these two pages by adding a *tab system*, through which the patient can switch between these two sites, since their nature is quite similar. Their similarity stems from the structure of the chat and feedback. The core of these two are the same, they are *threads* that have participants and hold messages. The core difference is the *answersheet id*. If a thread has a answersheet id equal to null, the thread is just a normal messaging thread. If, however, the answersheet id has a value, then it means, that this thread is connected to an answersheet and therefore acts as *feedback*.

4.6.2 Chat Overview

The chat overview will therefore list all the threads, that represent a normal chat. These chats will ordered in a list and show some information connected to it. They will show the subject of the thread, the name of the last sender and their last message. When clicking on this *list item*, we will be forwarded to the actual chat. Since the messages and threads will be loaded here, the newly implemented *message handler* will be heavily connected to this page.

4.6.3 Feedback Overview

The **feedback overview** shares many features with the **chat overview**. Here we represent the threads as cards, since the way we want to show the information is a little different. We have a higher focus on first message, that is attached to the *thread*, since this holds the feedback on the attached answersheet. Furthermore, we want to give the patient the opportunity to look at the answersheet, since he might have forgotten, what his answers where or what intervention it was. Lastly, when clicking on the card itself, the patient can access the chat to this feedback and ask questions to the feedback. Same as with the **chat overview**, a constant connection with the *message handler* will be needed to provide the data.

4.6.4 Chat

The **chat** is the communication point of the different threads. In this chat we load all the messages and present them to the patient. Own messages on the right, while other messages are placed on the left, following the common theme of *instant messaging*. Also following this theme, we have the *input field* and the *send button* as a footer and on top for the header, we have the *back button* and some information about the current chat. In order to send messages to server, we need a function located in the *message handler*.

4.6.5 Open and all Interventions

Since these two pages are mostly the same, we will design them concurrently. The core structure of both would be a big list of the available interventions, while the list for open one would be filtered. The interventions will we presented as cards, holding all the relevant information to inform the patient, what they represent. They will show the name and title of the intervention and further down will be the description. When clicking on them, more information will be presented and for the **open interventions** we will add an button, through which the patient can subscribe to them. To provide the data needed for

these pages, the intervention handler will be used. There we added functions to enable the fetching of needed information.

5

Implementation

In the following, we will go through the process of the implementation. We will start by undergoing the added components, show their UI and explain, how it came together and which connectivity to other components exist. Consequently, we will discuss the problems and challenges which came up in this process.

5.1 The new components

This section describes the added features and their pages. We will show screenshots in both the mobile view and also for desktop. Since android and iOS mostly distinguish themselves through icons and fonts, there is no need to separately inspect them. When talking about the pages, we will talk about their UI, how the navigation works and what you can access and do on the given site. Then we will explain the components needed to provide the content of the pages and elaborate on possible optimization and possibly existing problems that came with them.

5.1.1 New Home Page

The homepage, which on the first iteration of the project acted as a faster way to access the interventions of the patient, will now be changed to be like a Hub, which the patient can use to facilitate faster navigation. This is achieved by introducing card elements, which represent routes to the most used pages. These cards currently include following routes:

- My interventions
- My Profile
- Threads/Conversations
- All interventions

The Menu on the top left corner remains, however with the new pages added. The technical background of the new **homepage** has only really changed with the Ionic update, but more on that in section 5.2.

5.1.2 Threads

The new **thread page** combines the main part of the communication feature: The **chat** and the **feedback** system. On this page we can swipe between these two views and therefore choose, which service you want to use at the moment. This layout of two pages into the threads page is simply done by the *ion-tabs* template 5.1. Through this template we can assign the wanted pages into the routes of the module file and by setting them as its children, the router knows through the name tag in the *HTML* which page should be shown in which situation. Thus, on the left side, and as default starting location, we would find the **chat overview** and on the right, the **feedback overview**. The reason for this distinction is, that from a technical point of view messages for the chat and feedback are the same. If we have a thread with an *answersheet* attached, we know that the message is supposed to be considered a feedback. And vice versa, if there is not it is a normal conversation. Furthermore, we know that if an thread has more than the two participants, the thread is supposed to be a group. Since these two pages get loaded into the thread page and handle its own setup, not much more is needed in the thread page itself. Therefore, we move to the sub pages.

```
1  <ion-tabs>
2    <ion-tab-bar slot="top">
3      <ion-tab-button tab="chat-overview">
4  ...
5      </ion-tab-button>
6      <ion-tab-button tab="feedback-overview">
7  ...
8      </ion-tab-button>
9    </ion-tab-bar>
10 </ion-tabs>
```

Listing 5.1: The tabs layout for the overview pages

Chat-Overview

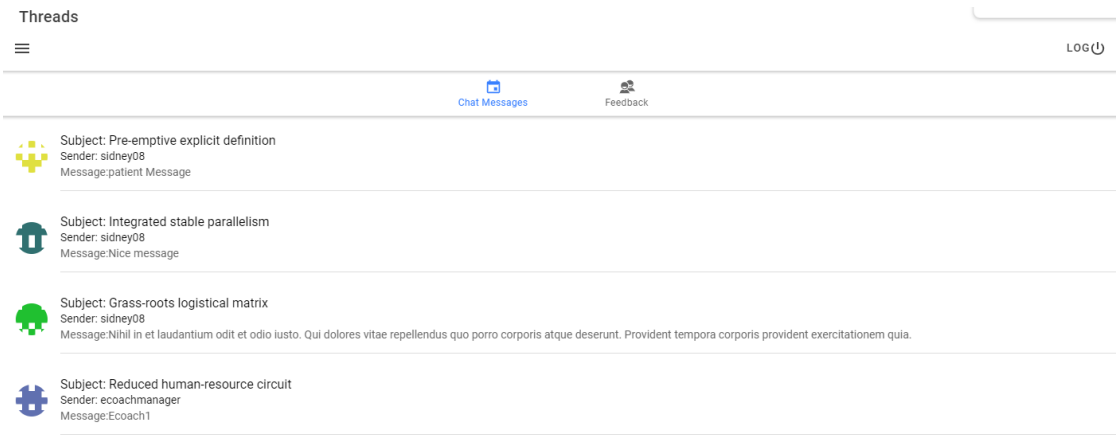

Figure 5.1: The chat overview as accessed via desktop

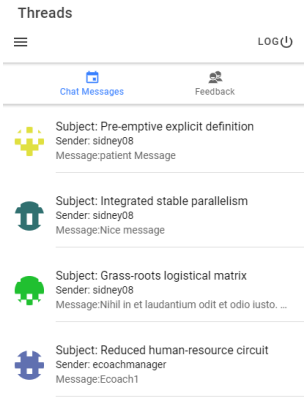

Figure 5.2: The chat overview on mobile

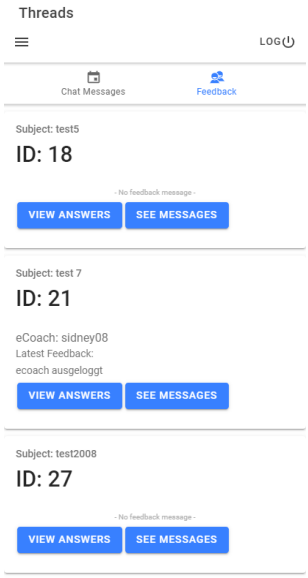

Figure 5.3: The feedback overview on mobile

This page, as shown in Figures 5.1 and 5.2, represents the top level of the chat. The core structure of it is a *ion-list*. It provides the different available chats and presents some info of each one.

```
1 <ion-list *ngFor="let thread of messageThreads">
2     <ion-item [routerLink]="['/chat/', thread.thread.id]" ...>
3       ...
4     </ion-item>
5   </ion-list>
```

Listing 5.2: The ion-list representing the chats

In line 1 of Listing 5.2 we can see the use of one of the many functionalities of angular, the usage of loops in our HTML while accessing items in our typescript file. As you can see, we loop through our *messageThreads* object, which holds all the threads that we know are a chat and generate an *ion-item* for each of these. These chat items for each thread therefore represent the *chatlike* selection menu of each conversation. One part of this list will be the picture of either the interlocutor or the group. On the right side the last sender name and the attached message that was sent will be presented.

Chat Page Since we are finished with the **chat overview**, we will now concentrate on the **chat pages** themselves. They are structured like we discussed it in our design Chapter 4. Since the mapping and layout of our approach, shown in Figures 5.4 5.5, is similar to other ones, the patient should not have a hard time to use it. On the top left we have the back button, which leads us back to the chat overview. The messages themselves are represented as the usual *chat bubbles*. They contain the name of the sender, the message and the timestamp, when the message was sent. Finally, on the bottom we find the *input field* for the message and on the left of it a button which sends the message.

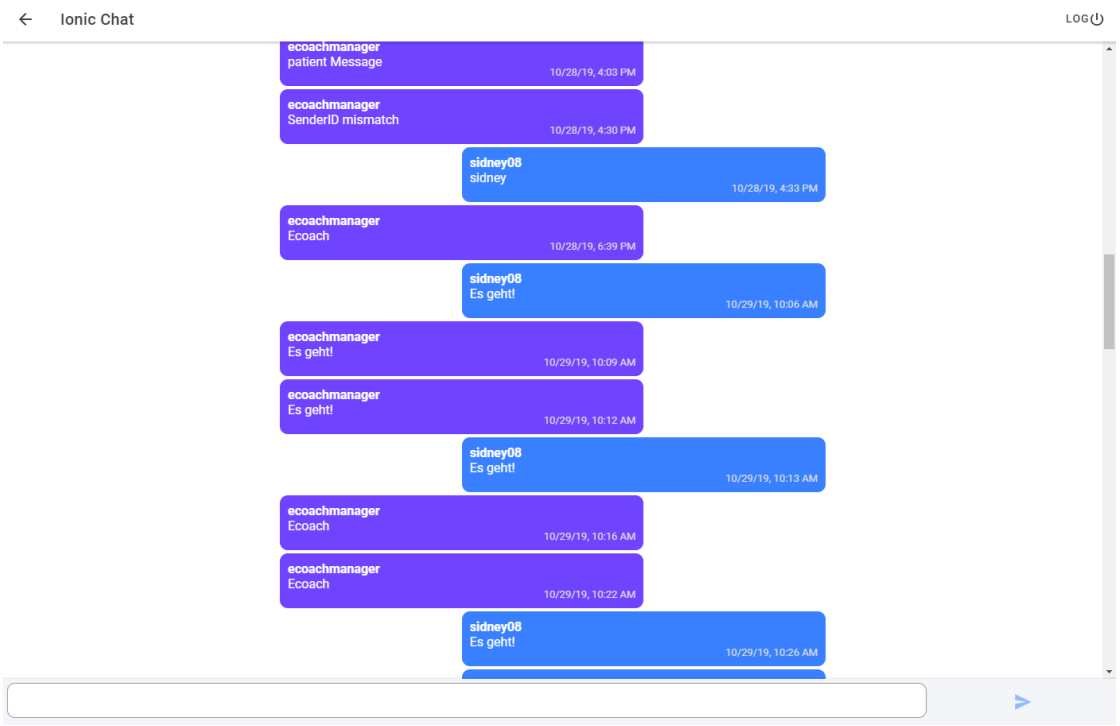

Figure 5.4: The Chat on desktop

Figure 5.5: The Chat on mobile

Before discussing the layout, we want to explain, how the chat gets its corresponding thread to receive data. We can pass the *thread id* to the page by adding an id to the *routerlink*. Once the module of a page has information, that said page can have an *id*, we can pass this *id* to the given page. To recall, in Listing 5.2 line 2, we can see the routerlink which includes the id of the *thread*. How we can read this id is shown in Listing 5.3.

```
1    this.threadID = this.activatedRoute.snapshot.paramMap.get('threadID');
```

Listing 5.3: The method to receive ids

Once the page gets initialized and calls this line, we can access the referred id. This id is needed, to load the *thread details* and therefore its messages. We can do this by accessing our *message handler* and using the *HTTP GET call* to load thread details by its id. An example is shown in Listing 5.4.

```
1  public getThreadDetails(threadID: number) {
2    return new Promise((resolve, reject) => {
3      let token = localStorage.getItem('token');
4
5      let headers = new HttpHeaders()
6        //.append('accept-language', localeLanguage)
7      let params = new HttpParams()
8          .append('limit', '0')
9          .append('token', token);
10     this.http.get(BACKEND_URL + '/api/v1/messages/threads/' + threadID, {
11       headers: headers,
12       params: params,
13       observe: 'response'
14     }).subscribe((response) => {
15       resolve(response);
16     }, (error) => {
17     ...
18     });
19   });
20 }
```

Listing 5.4: A snippet of the GET call

The native *HTTP* service of ionic is used, to make this call. We call the get method of this service and attach our modified URL with the thread id to it. To ensure, that we receive all data, we add *http params* like the token, to allow authentication and the limit of 0, through which all entries will be delivered.

```
{
    "data": {
        "type": "messages/threads",
        "id": "7",
        "attributes": {
            "subject": "Reduced human-resource circuit",
            "answersheet_id": null,
            "messages": 75,
      ...
      },
      "included": [
        {
            "type": "messages/messages",
            "id": "35",
            "attributes": {
                "body": "Est magni consequatur aut dicta corrupti.    Aperiam
    eveniet ut sed et dolores dolor.",
                ...
            },
            "relationships": {
                "author": {
                    ...
                    "data": {
                        "type": "users",
                        "id": "3"
      ...
}
```

Listing 5.5: The response from getting the thread list in JSON

We in turn receive a response containing *JSON*, in which our desired data is stored. With access to this JSON, a cut example accessible in Listing 5.5, we can read our messages of the thread and assign the *thread messages* to our message array. This message array then gets used in the HTML. Now back to the structure.

The main structure of the chat page is a grid and can be seen in Listing 5.6. We distinguish the messages by checking if the current message was sent by the user. If it

is, that means the patient himself sent the message and therefore want the message to be placed on the right side. If it is not, we will place it onto the left.

```
1  <ion-grid>
2      <ion-row *ngFor="let message of messages">
3
4        <ion-col size="9" sizeLg="8" sizeXl="4" offsetLg="1" offsetXl="3" *ngIf
       ="message.author !== currentUser" class="message other-message" >
5          <b>                    </b><br>
6          <span>                     </span>
7          <div class="time" text-right><br>{{ message.createdAt | date:'short'
       }}</div>
8        </ion-col>
9
10       <ion-col size="9" sizeLg="8" sizeXl="4" offset="3" offsetLg="3"
       offsetXl="5" *ngIf="message.author === currentUser" class="message my-
       message">
11         <b>                    </b><br>
12         <span>                     </span>
13         <div class="time" text-right ><br>{{ message.createdAt | date:'short'
        }}</div>
14       </ion-col>
15
16     </ion-row>
17   </ion-grid>
```

Listing 5.6: The structure of the chat

When sending a message, we use the name of the current user, the message that was typed into the *input field* and the thread id to send the message to the server. This will be handled in the *message handler*, in which we can pass this information and make a *POST* call to the *API*. At the same time, we need to display our message in the chat. To do this, we generate a new message object and set the sender and message into it. After this, we simply push the message into our message array. Therefore, the loop in our HTML will add this message to the view.

As discussed in Chapter 4, to enable live chat, we load messages every 10 seconds and check for new ones. Once entering the chat page, we lower this interval to 3 seconds,

to allow a faster interaction. In Chapter 6 we will talk about the possible integration of the *external push-notification service*. If a new message is available, this message will simply be added to our message array and therefore be pushed into the view. In the Section 5.1.3, we will explain more thoroughly how this refresh is done.

Feedback-Overview

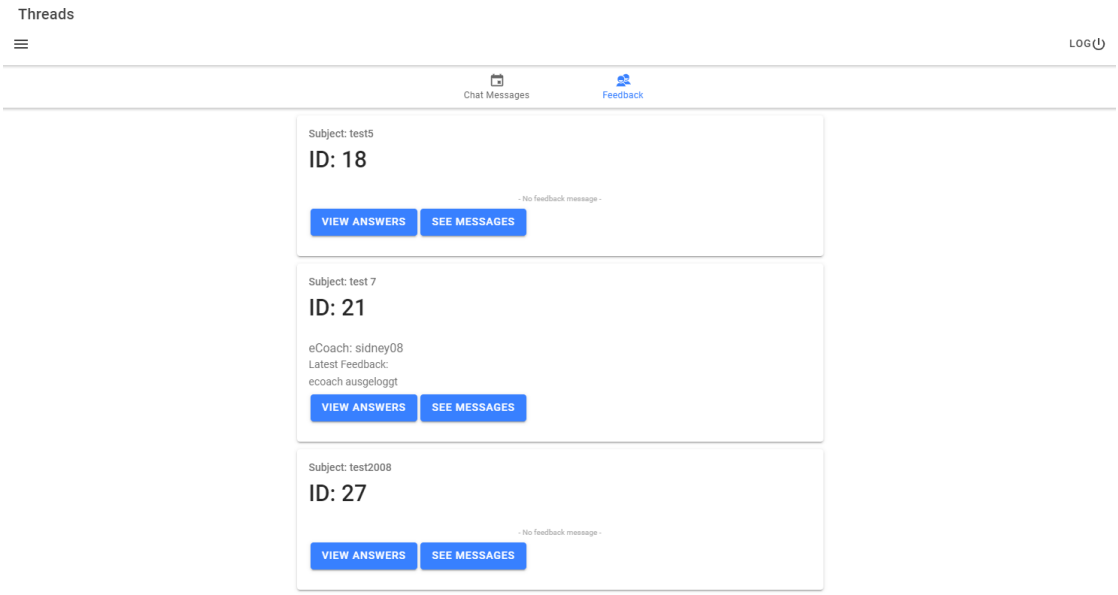

Figure 5.6: The Feed-Overview as accessed via desktop

Since the core structure is the same for chat and feedback, many principles of above also apply here. On this feedback overview page, shown in Figure 5.6 and 5.7, we display the different feedbacks, this time however as cards, since the use case is a bit different than the chat. In these cards we display information like the subject name, the eCoach that left the feedback and the message attached to it. Furthermore, since the patient might not remember the intervention attached to this feedback, we give the opportunity to view his answersheet and recall. The other information we want to enable is to also go into a

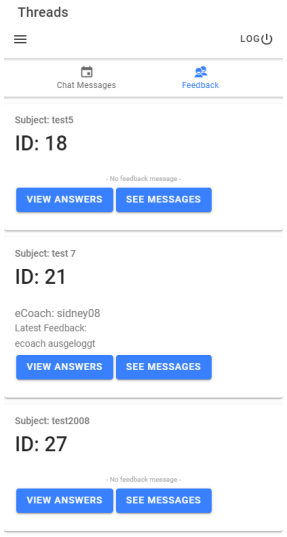

Figure 5.7: The Feed-Overview on mobile

chatlike page, in which the patient can get more details and ask questions about that feedback.

The way to do this is mostly the same than for the chat overview. We load the messages, filter them by its answersheet id, push them into the list and display them by iterating through these entries and generate the according cards. The *show chat button* is therefore also the same routerlink as in the chat. We provide the thread id and load the messages or in this case feedback. To enable the view of the answersheet and also the show name of the intervention etc. a few more loading steps are needed. Since all we have to work with to get the information is the answersheet id attached to the thread, we have to get the information by searching for it in the provided components. We start by inspecting, which *questionnaire id* is attached to the answersheet, use that to get details for said *questionnaire* followed by the acquiring the connected *study id* which in the last step can be utilized to obtain its name and other data. After going through this cycle we can in turn set the router to the lesson page with the *lesson/questionnaire id*, attach our answersheet and show the input of the patient on that lesson. Since the feedback page is basically the chat page, we will not go into detail for it.

5.1.3 Notifications

As discussed in the design, we differentiate between message and reminder notifications. The following section explains the message notifications.

Message Notifications

To introduce the *message notifications*, we had to implement a method, that pulls messages every ten seconds. In this method, which has access to the threads, we can check how many messages are currently inside this thread and compare this value to the number of *pulled messages* for the thread. If we have more messages in our pull, we want to append the number of messages to our thread messages that are missing. In this condition, we will then call the *notification function* in our *notification handler* and show a notification. Therefore we could show the patient, if he got new messages in this time span as seen in Figures 5.8 and 5.9.

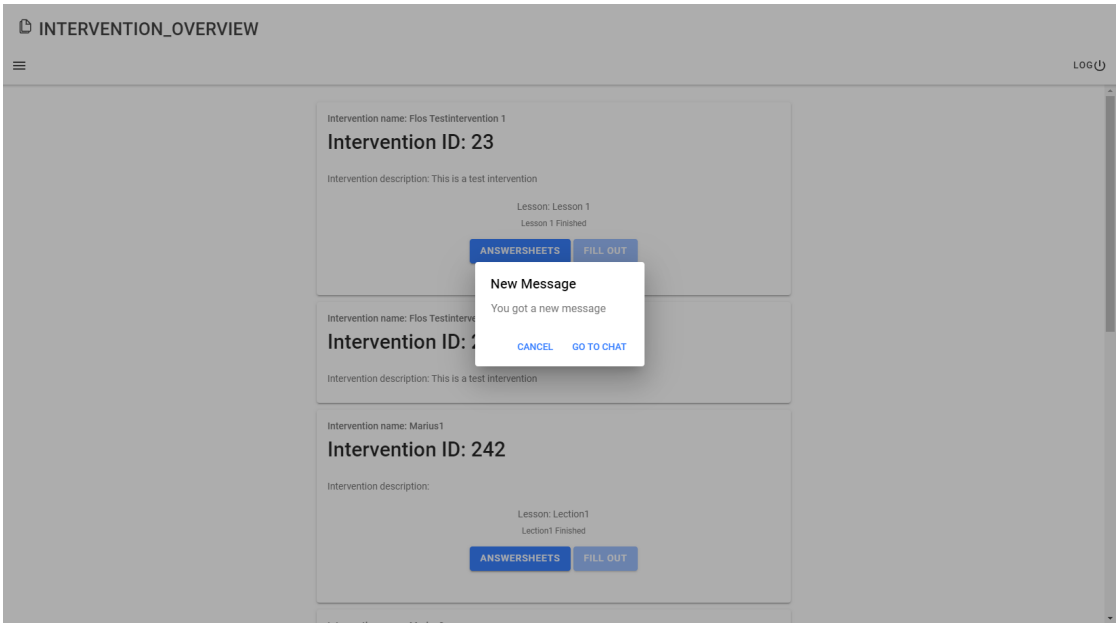

Figure 5.8: A message, that informs the patient of new message as seen on desktop

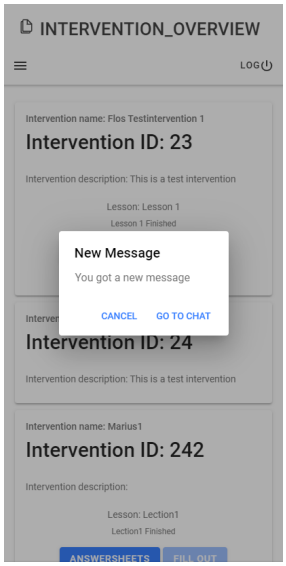

Figure 5.9: A message, that informs the patient of new message as seen on mobile

For notifications, that have to trigger at certain points, like the reminder for interventions, we will use a different method. This type of notification can be added, by first checking for the platform and if the platform is either android or iOS, we can use the *local notifications service* provided by Ionic. Since this is a *cordova* restricted function, we do not have it yet available for the desktop. Similarly to above, we would not want to send a tray message, if the patient is currently using the app. We can handle this, by simply listening for the *local notifications* and, if one gets triggered, transform this notification into an *popup* and show it inside the app itself.

5.2 Challenges and Problems

Here, we will state the challenges and problems related to the implementation. Starting off with the difficulty of the UI between platforms and continuing with the Ionic upgrade.

5.2.1 UI between platforms

A few issues regarding the UI of our cross-platform application and its different platforms occurred in this iteration. Because of the way ionic is built, smartphone and web users use the same core structure, even though in some cases a differentiation would be desirable. As analyzed in the standards of instant messaging in Chapter 4, it would be desirable to combine the chat overview page with the chat. However, based on time constraints, the separation of different core structure between the platforms has not been fully implemented. The solution for this would be, to introduce a *split pane* in the threads page, which collapses, if the screen size is not wide enough. If it is, then the *splitscreen* will show and therefore display the chat overview on the left and the chat on the right.

5.2.2 Upgrade to Ionic 4

On the point as the patient app was started, the current version of Ionic was 3. In January of 2019, Ionic 4 was introduced and with it, many changes, improvements and refinements of the whole Framework. While the app worked under Ionic 3, there where some problems regarding the emulator, the navigation process and the CLI. This combined with the performance upgrades coming with these changes demonstrated, that the application should be updated before starting the implementing process of the communication module. However, this update from Ionic 3 to 4 was not as trivial as running an update command. Because the changes were fundamental on the core the project needed to be carefully adopted to Ionic 4. As a helper, we used the migration guide of Ionic [23]. This was done, by first generating a new ionic project with the newest Ionic version installed. This in turn generated a Ionic 4 project. This newly setup project then needed to be enhanced by adding all the *pages, services, components and models* through the CLI in order to have the permission, access and dependencies needed. Once having the structure of the project back, we needed to start importing the code into the new project. This was done in a systematical way: Starting with the pages, their typescript files and HTML, then the providers and then the components and models. Once we had everything in the right place, the code needed to be adjusted to remove error. We had to remove all imports and redo them since the names and paths of some

have changed and we needed to redo them all to ensure correctness. After that we would change the structure of the HTML to use the proper tags. We will now present few of these changes.

Listing 5.7 shows, how a button was setup in Ionic 3, in Listing 5.8 the new method is shown. We can observe, that the tag changed removing the need to add an extra parameter to the button. When having a button with an link, we used the *a* tag, while we can now just attach the *href* to the *ion-button* directly.

```
1  <button ion-button (click)="do()">
2    Default Button
3  </button>
4
5  <a ion-button href="#">
6    Default Anchor
7  </a>
8
```

Listing 5.7: The button in ionic 3

```
9  <ion-button (click)="do()">
10   Default Button
11 </ion-button>
12
13 <ion-button href="#">
14   Default Anchor
15 </ion-button>
16
```

Listing 5.8: The button in ionic 4

Similarly in Listing 5.9 and 5.10 the tag for the menu button changed.

These and any more syntactic changes have been made and needed to adapted. Once all these were adjusted, we had to remove the *navigation control system* of older Ionic versions for the the *routing*. This is probably one of the biggest changes. With earlier versions, ionics navigation worked by having a main *navigation controller* and when navigating to another page, we could push or pop the pages together with navigation parameters that could be used in the new pages. In Listing 5.11 we can see, how the navigation used to work and in Listing 5.12 how the attached parameters could be read.

```
17 <button ion-button menuToggle>
18   Toggle Menu
19 </button>
20
```

Listing 5.9: The menu button in ionic 3

```
21 <ion-menu-toggle>
22   <ion-button>
23     Toggle Menu
24   </ion-button>
25 </ion-menu-toggle>
26
```

Listing 5.10: The menu button in ionic 4

```
this.navCtrl.push(OtherPage, {
      id: "123",
      name: "Carl"
    });
```

Listing 5.11: The navigation control of Ionic 3

```
class OtherPage {
  constructor(private navParams: NavParams) {
     let id = navParams.get('id');
     let name = navParams.get('name');
  }
}
```

Listing 5.12: Ionic 3 and its method to read the navigation parameters

Now we use the method, that is the standard with angular. With this routing, we also have a main container, the router outlet, in which we can load all the pages and navigate in them. In order to use them, you would have to declare them in the module file of the page and then access them directly in the HTML by using the routerlink and the path. In Listing 5.13 this method by setting the routerlink to the desired page, if needed with an id attached, as we saw in Listing 5.2, is shown.

```
<ion-button routerLink="/dashboard"
  Login
</ion-button>
```

Listing 5.13: An example for the router with Ionic 4

One problem with this routing change was, that because the navigation parameters could not be used to transfer objects like the whole lesson model anymore, we had to think of a system that works smoothly with only the ids. The solution was to pass the *lesson id*, then either loading the lesson from the server or searching for it locally and set the lesson through this technique on the pages as, again, seen in Listing 5.2.

Positives and negatives of the Update The positives of this update where many fixes inside the *CLI* that persisted in ionic 3. The application also is faster and builds

more stable than before. With the addition of the angular routing we also had a more used and therefore more maintained way for navigation. In general, the newer version of angular allowed new methods in some cases and is more resilient.

Negatives include, that the update took quite a lot of time and is currently missing the translation module. One reason for this delay, as explained shortly above, is because some parts of the project had to be redone completely different than in the first iteration of the app.

Not really associated with the update but also noteworthy is the abandonment of the plugin *secure storage*, since this plugin is no longer maintained.

6

Discussion

In this section, we want to discuss the work in general. We will state the significant problems when working with the project and what did not work as smoothly as one would like. After summarizing, how the push notification feature could be implemented, we will check the requirements set in section 3 and inspect, which were fully implemented and which not.

6.1 My own difficulties

Since this was the first project of that size I ever worked on, especially the beginning was quite hard. Getting to know all the terms, how they are connected, how the systems works, the structure and the *API*. In addition, while i worked with *Javascript* before, with *Typescript* being the superset and its heavy use of *await* and async calls together with a lot of *promises*, I had some real problems in these regards. This together with to inter connectivity of *Angular* and *Ionic*, there were some complicated use cases and methods, I could not quite get a grasp of.

Lastly, my biggest problem in the project, *Firebase*. Firebase is the *external notification feature* that we would like to use in our system. I tried many hours, to add this feature to our project, however I could not get it to receive the messages, that the server would send, once a message gets sent by another participant in a thread. As a section, we will discuss, how the implementation of Firebase should theoretically be done.

The other big delay in the project was the *Ionic upgrade*. While it was the right decision to do, it created a mountain of work in a chain reaction. First, a lot of the styling that was

done before had to be redone, some technologies were not supported anymore, with the lack of navigation parameters the whole system of transferring lessons from one page to the other had to be reevaluated and lastly reformed. All these problems made parts of the system deprecated and had to be redone.

6.2 Future implementation of an external notification service

When researching, a couple of these notifications services were found. There was Firebase [24], Onesignal [25] and Parse [26]. For our requirements, Onesignal was not really an option, because information sent through their service in the free version is not encrypted and therefore not safe, even with our vague messages we would not want the name etc. to be seen. Parse is a good service, but the push notification service that lies under the hood is mainly consisting of Firebase. For these reasons we settled for Firebase. With its messaging service, we had everything that we needed and its freemium model fits the requirements. They are GDPR-ready [27] and offer some more services, that could be beneficial to our system later on.

When using Firebase with Ionic however, there were more possibilities to use it. There would be AngularFire [28] or the native integration of Firebase to Ionic, which relies on the cordova plugin Firebase [29]. Furthermore FirebaseX [30] exists, which is as, explained on the Ionic documentation, a maintained fork from the unmaintained ionic-native plugin from above. Lastly the cordova plugin fcm [31] and similarly cordova plugin fcm with dependency updated [32].

Since the project will persist for some time, the chosen method should be a maintained one. Most plugins are not natively maintained by the Ionic team themselves, but more projects of independent people. In order do have Firebase available on cordova and consequently on our mobile platforms, we have to use a cordova plugin. On the browser platform, Angularfire is a wise choice, since it is actively maintained by the angular team. Therefore the choice for mobile has to be between the cordova plugin Firebase, FirebaseXs implementation for it and the cordova plugin fcm with dependecy updated (Since the cordova plugin fcm has not been update since January 2018).

Possible intergration Before adding Firebase to our Ionic project, the project has to be enhanced to be a progressive web app (PWA). The main idea for this is, that it enables features like sending push notifications and having a service worker, which can listen to incoming messages even if the app is not actively used and more.

The core of how it works is quite simple: By making our app a PWA through the command

```
1 ng add @angular/pwa --project *project-name*
```

Listing 6.1: The command to allow the application to be a PWA

Through this, we get the ability to use service workers and get our manifest.json/ manifest.webmanifest. Service workers act as small script that run in the background and enable different features like our desired push notifications, but also enable the use of some hardware like sensors of smartphones and more. So this service worker can handle, cache and block different data streams from and to our application.

Since the notifications of the Firebase could and will be sent, even if the patient is not actively using the app, this is a crucial addition to enable this feature for desktop users. Since this is a natural feature of the mobile platform, we can use native features for them. Since the notifications of Firebase and our native notifications are a bit different, we will add an independent component for it. Namely, the fcm service (Firebase Cloud Messaging), where all the handling of foreground messages will be located. This FCM handler, will therefore be responsible for incoming notifications sent by Firebase. Here we will initialize the service worker and start the observable, that listen for these messages. This service worker will, in contrast, administer the notifications when the app is in the background.

6.3 Requirements Comparison

In this section we want to go through all of our requirements declared in Chapter 3 requirements. We try to explain, how successful each of them was implemented and, if not, elaborate why and which parts have not.

Requirement ID: 1, Title: Chat This Requirement is fully functional. The patient can access a chat, send messages in it, read the messages of the other participants and the messages get updated every 10 seconds. Therefore the messages will constantly refreshed and are up to date.

Requirement ID: 2, Title: Basic Feedback Function The Basic Feedback function works as well. The patient can access the feedback page and in it, look at his current feedback, look up the answersheet and ask questions upon the feedback in the attached chat.

Requirement ID: 3, Title: Enhanced Feedback Function Since this Requirement is not currently possible, since the eCoach system does not currently support this feature, we have not implemented it. Once the feature is accessible, the needed implementation is not too complicated. The breakpoint has to be read out, at which the intervention will stop and therefore show a popup and send the patient back to the homepage. Once the eCoach gave feedback, the feedback can be accessed and through this page, a router to the rest of the intervention could be set.

Requirement ID: 4, Title: Notification service This requirement, only works for messages and feedback, when actively using the application and for the reminder. If the app registers a new message/feedback, the patient will be informed via a popup, that a new message/feedback is available. The other notifications have not been implemented in the backend yet and therefore do not work in this iteration.

Requirement ID: 5, Title: Push-Notifications As discussed above, this does not work as desired, because our external notification service was not implemented. While the user can get notifications, when the user wants to be reminded, however the active pushing of notifications only works on smartphones.

Requirement ID: 6, Title: Remember me function This feature works and can be activated on a desired intervention when clicking the icon. At this point the user can select when he wants to be reminded. This feature however is as well limited to smartphone users.

Requirement ID: 7, Title: Watchlist Based on the priority, this feature has not been implemented

Requirement ID: 8, Title: List of all interventions This requirement is implemented and allows the patient to see all interventions stored in the system.

Requirement ID: 9, Title: List of open interventions This requirement is implemented and allows the patient to see and subscribe to the open interventions that are available in the system.

Requirement ID: 10, Title: Filter function for interventions Based on the priority, this feature has not been implemented, however would be a useful addition to the app.

Requirement ID: 11, Title: Basics of conditional prompts This feature was not implemented. Based on the time constricts, mainly the chat and notification requirements were prioritized. At the end, no time was left to implement this feature.

Requirement ID: 12, Title: Improving upon the conditional prompts Since the basic feature of this has not been implemented, this feature did not get included as well. With its priority, this feature would not have been implemented nonetheless, since it will be a part of another iteration.

As we can see, the main part to introduce the communication module has been implemented, even though with some missing add-ons. While the patient can communicate with the eCoach and other patients without any problems, whenever he is not actually

using the application and looking at the chat overview, he would not know, since the push notification service does not work.

7

Conclusion

In this last section, we will summarize the work and review the lessons we learned in the process of working with the project. Finally, a future outlook regarding the project will be given, which aspects could be improved and what will come next.

7.1 Summary

Now to recap everything. In the fundamentals, we discussed the Internet and mobile based interventions, how its positives could improve upon the current shortage of therapists and waiting time, discussed the difference and efficiency of guided and unguided interventions for certain mental health problems and therefore clarified, how our introduction of the communication module would help the application. Following, we introduced our requirements and described the features that we would like to add with certain priorities. Hereafter, we went through our design process. We illustrated, how the structure would look like and how the components will interact with each other, how the UI will look like and which components and pages will be added in which way into the system. Consequently, the implementation phase was described. In here, we started off by naming the new and upgraded components, then went through our UI and UX and how it connects to the other components while operating. Finally we talked about the challenges and problems that occurred during the process, mainly the Ionic upgrade. After that, the discussion on the whole process was started. We wanted to explain, how some problems occurred, what my own problem during the project were and for future reference, we explained how the external notification service Firebase could be implemented into the system, as well as a small introduction to the conditionals.

Concluding this, we went through the requirement comparison and explained, how each of them has or has not been successfully implemented into the system.

7.2 Lessons learned

I am really grateful, that I could work on this system. While I had my difficulties, accessible in Chapter 6, the technology of cross platform and Angular in particular is very interesting and I would like to learn more about it. The concept, of fetching data through the promises as await and async calls is really interesting, which parts are available when and how you can use the fetched data effectively in your code. As an extra, the insight into progressive web apps (PWA) that I got through Firebase is also fascinating. Being able to implement a app, or Web app, with close to the same possibilities on a native android app and enabling some of these features on the desktop browsers is truly captivating.

7.3 Outlook

In retrospective, while keeping our problem statements in mind, we can inspect, that the application now includes a communication module. Through this module, the app is now capable of sending messages to groups, other patients and the eCoach. He can look up and receive messages, gets notifications for them when using the app and is therefore able to communicate. Moreover, through this module, the patient can now access feedback for certain interventions. This feature consequently allows the project and the app to utilize guided interventions. This is beneficial for patients with particular mental health disorders and thereupon allow an efficient treatment for these patients as we inspected in Chapter 2.

Furthermore, as discussed in Chapter 6, the project will continue and be further expanded upon. The next steps will be, to integrate the Firebase service and include the conditionals into the application. After that, the notification system of the whole project will be discussed and in a later iteration fully introduced. Another crucial aspect will be

to improve the desktop experience, since, through the nature of Ionic, most pages are mainly optimized for a smartphone experience.

Bibliography

[1] : Study waiting times in germany. Website (2019) https://www.bptk.de/wp-content/uploads/2019/01/20180411_bptk_studie_wartezeiten_2018.pdf; Last Accessed 23.10.19.

[2] : American psychological association. (2019, march 15). mental health issues increased significantly in young adults over last decade: Shift may be due in part to rise of digital media, study suggests. Website (2019) https://www.sciencedaily.com/releases/2019/03/190315110908.htm; Last Accessed 23.10.19.

[3] : Mental health issues on the rise among adolescents. Website (2019) https://www.ajmc.com/focus-of-the-week/mental-health-issues-on-the-rise-among-adolescents-young-adults; Last Accessed 23.10.19.

[4] : Goodtherapy shortage of health professionals. Website (2019) https://www.goodtherapy.org/blog/psychology-facts/is-there-shortage-of-mental-health-professionals-in-America-0308197; Last Accessed 23.10.19.

[5] : Lieferando. Website (2019) www.lieferando.de/app; Last Accessed 18.10.19.

[6] : Uber. Website (2019) www.uber.com/de/de/ride/; Last Accessed 18.10.19.

[7] : One example of making doctors appointment online. Website (2019) https://home.cgm-life.de/portal//#/appointment/?institution=17dfe010-3ec9-4809-ae3c-ac499d38e47f; Last Accessed 23.10.19.

[8] : One example of an online doctor. Website (2019) https://www.teleclinic.com/; Last Accessed 23.10.19.

[9] Weisel, K.K., Zarski, A.C., Berger, T., Krieger, T., Schaub, M.P., Moser, C.T., Berking, M., Dey, M., Botella, C., Baños, R., Herrero, R., Etchemendy, E., Riper, H., Cuijpers, P., Bolinski, F., Kleiboer, A., Görlich, D., Beecham, J., Jacobi, C., Ebert, D.D.: Efficacy and cost-effectiveness of guided and unguided internet- and mobile-based indicated transdiagnostic prevention of depression and anxiety (icare prevent): A three-armed randomized controlled trial in four european countries. Internet Interventions **16** (2019) 52 – 64 SI:ICare – Horizon2020.

[10] Harrer, M., Adam, S.H., Baumeister, H.E.a.: Internet interventions for mental health in university students: A systematic review and meta-analysis. (2018)

[11] Domhardt, M., Geßlein, H., Baumeister, H.E.a.: Internet- and mobile-based interventions for anxiety disorders: A meta-analytic review of intervention components. (2018)

[12] Boß, L., Lehr, D., Berkin, M.E.a.: Evaluating the (cost-)effectiveness of guided and unguided internet-based self-help for problematic alcohol use in employees - a three arm randomized controlled trial. **15** (2015)

[13] Young, C., Campbell, K.: Guided Versus Unguided Internet-Delivered Cognitive Behavioural Therapy for Major Depressive Disorder and Anxiety Disorders: Comparative Clinical Effectiveness. (2018)

[14] : Real cost of untreated mental illness in america. Website (2019) `https://www.constellationbehavioralhealth.com/blog/the-real-cost-of-untreated-mental-illness-in-america/`; Last Accessed 20.10.19.

[15] : World health report. Website (2019) `https://www.who.int/whr/2001/media_centre/press_release/en/`; Last Accessed 20.10.19.

[16] : National institute of mental health. Website (2019) `https://www.nimh.nih.gov/health/statistics/mental-illness.shtml#part_154788`; Last Accessed 20.10.19.

[17] Andersson, G., Titov, N.: Advantages and limitations of internet-based interventions for common mental disorders. World Psychiatry **13** (2014) 4–11

[18] : Psychological intervention wikipedia. Website (2019) https://en.wikipedia.org/wiki/Psychological_intervention; Last Accessed 23.10.19.

[19] Baumeister, H., Reichler, L., Munzinger, M., Lin, J.: The impact of guidance on internet-based mental health interventions — a systematic review. Internet Interventions **1** (2014)

[20] : Moscow method. Website (2019) https://en.wikipedia.org/wiki/MoSCoW_method; Last Accessed 18.10.19.

[21] : Whatsapp. Website (2019) https://www.whatsapp.com/?lang=de; Last Accessed 18.10.19.

[22] : Telegram. Website (2019) https://telegram.org/; Last Accessed 18.10.19.

[23] : Ionic migration. Website (2019) https://ionicframework.com/docs/building/migration; Last Accessed 18.10.19.

[24] : Firebase. Website (2019) https://firebase.google.com; Last Accessed 18.10.19.

[25] : Onesignal. Website (2019) https://onesignal.com/; Last Accessed 18.10.19.

[26] : Parse. Website (2019) https://parseplatform.org; Last Accessed 18.10.19.

[27] : Dataprotection gdpr. Website (2019) https://eugdpr.org; Last Accessed 18.10.19.

[28] : Angularfire documentation. Website (2019) https://github.com/angular/angularfire; Last Accessed 18.10.19.

[29] : Plugin for cordova firebase. Website (2019) https://github.com/arnesson/cordova-plugin-firebase; Last Accessed 23.10.19.

[30] : Plugin for cordova firebasex. Website (2019) https://github.com/dpa99c/cordova-plugin-firebasex; Last Accessed 23.10.19.

[31] : Plugin for cordova including firebase cloud messaging. Website (2019) `https://github.com/fechanique/cordova-plugin-fcm`; Last Accessed 23.10.19.

[32] : Plugin for cordova including firebase cloud messaging with updated dependecy. Website (2019) `https://github.com/andrehtissot/cordova-plugin-fcm-with-dependecy-updated`; Last Accessed 23.10.19.

List of Figures

Name: Marius-Lukas Ziegenbein Matriculation number: 909614

Honesty disclaimer

I hereby affirm that I wrote this thesis independently and that I did not use any other sources or tools than the ones specified.

Ulm, 19.11.2019 .....

Marius-Lukas Ziegenbein