ulm university universität uulm

**Facility of Ingenieurwissenschaften, Informatik und Psychologie**
Institute of Databases and Information Systems (DBIS)

# Design and Implementation of a Cross-Platform Application for Internet- and Mobile-Based Interventions

Bachelor Thesis at Ulm University

**Submitted by:**
Florian Matthias Haschka
florian.haschka@uni-ulm.de
870351

**Reviewer:**
Prof. Dr. Manfred Reichert

**Supervisor:**
Robin Kraft

2019

Version April 5, 2019

Satz: PDF-$\LaTeX\,2_\varepsilon$

# Contents

# 1 Introduction

This chapter will give a small introduction on what the thesis and the implemented application is about. This covers an explanation for the application's motivation and the environment the application will be used in.

## 1.1 Motivation

For internet based interventions (IBIs, 2.1) having noticeable impacts on the mental health of patients, the usage of these IBIs is expanding. Following this trend, the Institute of Psychology and Education, Department of Clinical Psychology and Psychotherapy at Ulm University (KLIPS) want to use these for research purposes. Offered tools of other suppliers do not fit for the use of KLIPS as they are lacking needed features and flexibility. They either do not provide the ability to create and use own interventions, are expensive to use or do not provide enough flexibility as needed by KLIPS. Hence, the goal is to create a software system covering the needed requirements to offer the ability to support the institute's research. To provide even easier access to IBIs for patients, the software system should follow the concept of Internet and Mobile based Interventions (IMI, 2.1). The outcome expected in terms of user acceptance and results of the treatment are even higher due to the usage of the IMI concept.

For this purpose, a plan for a software system was created to provide the support described above. The software system should offer the ability to create and manage IBIs and to guide and observe the patients participating in them. For easier development and handling, the system was planned to be split into four different parts, each of them described more precisely in Section 1.2. Each part will be developed separately but all parts will work together to build one big functional software system.

## 1.2 Use-case

The system should consist of following parts: the Content Management System (CMS), the back-end server, the E-Coach platform and the Patient-Module, with the last mentioned part being the topic of this work. This chapter will cover the tasks of these parts and how they will work together. For better understanding of the Patient-Module's tasks inside this software system, each part will be described below.

The first part is the Content Management System (CMS). This browser website allows its users to create and manage interventions and their corresponding lessons. Its task is to create and manage interventions used by the software system. This module will only be available to editors. These are a group of people having the right to create and manage interventions and will most likely be members of KLIPS.

The E-Coach platform is responsible for providing all functionality e-coaches need to guide their patients through their interventions. E-coaches are a group of people authorized to accompany patients through their therapy. This usually requires a medical education since all data collected of users is private and further actions might concerning the patients treatment might be needed. Just as the CMS, the E-Coach platform is a browser-based website. The platform allows e-coaches to handle the interaction with their patients and to assign and invite patients to interventions.

The Patient-Module embodies the patients' access point to the sytsem. It shows patients their interventions and provides all functionality to be able to participate in them and provides the ability to interact with the corresponding e-coach. This module can be used by any kind of user.

The back-end is a web-server providing all data used by the other system parts and linking them together. All data created by the three other parts is stored on the back-end and exchanged through it. Taking the current planning for the software system into account, the CMS, E-Coach platform and the Patient-Module will not directly communicate with each other but will communicate with the help of the back-end. The data will be send to the back-end and from there to the corresponding platform.

Given this architecture of the software system, the application described in this work will require the other parts to work as intended since it will need data provided by the CMS and the guidance and administration of the E-Coach platform. Assuming that these requirements are met, the application will interact with the other parts through the back-end by loading the corresponding information, providing the ability to participate in interventions and send created data back to the back-end. All data used by the application is provided by the back-end. This whole process can be observed by the corresponding e-coach using the E-Coach platform.

# 2 Fundamentals

For better understanding of the thesis, the following will define and explain fundamental information for for the thesis.

## 2.1 Internet Based Interventions And Internet And Mobile Interventions

Internet based Interventions are a psychological treatments with a relatively short history [1]. The concept of IBIs is that patients regularly login to a platform in form of a website or any other form of online application, which provides a series of online modules, further referred to as lessons. These lessons usually have to be completed in a given time period, belong to a corresponding intervention and completing them unlocks following lesson. Usually, these lessons include questionnaires concerning the personal problems of the patients. These problems refer to the needed medical treatment for the patient. For example, a patient might suffer from tinnitus. IBIs provided for him will most likely include questions about this specific illness. This provides better understanding of the patient's condition for the e-coach and can help finding better fitting treatments. This is also expected to help the patient directly by getting patients to confront themselves with their illness.

IBIs can be divided into two categories, which can be distinguished by lessons being guided or not. Guided lessons include contact to a therapist who will be embodied by e-coaches in this software system. This implies mainly the guidance given through the e-coach's involvement by adding the patient to new interventions and giving feedback for finished lessons. The amount of time, which is needed by the therapist for this guidance, depends on the intervention itself.

As access to basic health-care is not as widely spread as expected [26], IBIs provide the ability to reach out to those who do not have access to health-care. Providing anonymity and easy accessibility through the internet, this concept offers a suitable alternative for clients with psychological problems to seek out for help or do not want to see a doctor in person. Research has shown that the outcome of IBIs is comparable to those of face-to-face therapy [26]. Possible gains could be examining clearer clinical profiles of patients and even evaluating other models of therapy.

The major advantages gained from IBIs are the possibilities to provide anonymity for patients that might not be ready to visit therapy. Furthermore, IBIs are way more cost-efficient than face-to-face therapy as a result of therapist guidance not being needed compulsorily and taking less time if needed. Additionally, therapists can observe the progression of patients more easily throughout their participation in IBIs.

In addition to IBIs, Internet And Mobile Interventions are used to extend the possibilities of IBIs. IMIs follow the same principle as IBIs, but extend the platform for online interventions to mobile devices. This can be achieved through SMS or mobile applications provided for users. One example for the achievements provided through IBIs can be viewed in [17]. By providing IMIs for patients with depression, beneficial effects on the patients could be detected.

## 2.2 The Concept Of Cross-Platform-Applications

Before being able to understand the design of the application, the concept of Cross-Platform-Applications has to be explained, since the application follows the idea of this concept.

> Cross-platform software is software that is designed to run on more than one combination of hardware and software. For example, Adobe Creative Cloud and Microsoft Office 365 can run on macOS and Microsoft Windows. - [19]

Usually, different platforms require different implementations of the application since each platform uses its own application programming interface (API), thus forcing the developer to implement the same application multiple times, each version fitting for one platform. This process is resource- and time-consuming as implemented systems cannot be reused on other platforms.

One way of reducing the amount of time and resources needed to implement a Cross-Platform-Application is using Cross-Platform Frameworks.

> Frameworks are a way of delivering application development patterns to support best practice sharing during application development [...]. A framework is a reusable design expressed as a set of abstract classes and the way their instances collaborate. It s a reusable design for all or part of a software system. - [16]

Following the message of this statement, frameworks provide application-specific software to provide a standard way to build and deploy applications. In other words, they assist the developer in developing software by providing additional functionality. As the name already describes, Cross-Platform Frameworks provide functionality to make the application work on different platforms.

The essential property of these frameworks is that the application only has to be implemented once for all platforms. This does not mean that the software will run on all existing platforms. Rather, frameworks define the platforms on which the software will run. This is usually achieved by different deployment of the application for each platform, which is done by the framework. For the software project introduced in this thesis, the Cross-Platform Framework Ionic was used. The framework will be introduced more precisely in Section 4.2

Summarizing, Cross-Platform-Applications are software products working on a specified number of specified platforms, usually realized through Cross-Platform Frameworks, that enable the implemented software to run on different platforms without the need to implement the application for each platform separately.

Although taking off work for the user, cross-platform applications usually restrict the possibilities for implementation as it has to run on all defined platforms. For this purpose, the framework has to make compromises in terms of available functionality. While implementing on native languages for a platform, the developer can use all available functionality offered by the programming language and the framework as it will work guaranteed. By trying to implement functionality for multiple platforms whose functionalities differ, the developer has to take into account that the implemented functionality might not be available on all platforms. Implementing functionality not supported on a platform might either result in the framework not accepting the source code or in the occurrence of errors on the concerned platform. Furthermore, this condition might lead to unexpected behaviour or even to failure of the application. This is especially important for applications running both on mobile as well as on desktop or browser. Mobile devices may have more possible functionalities accessible by a cross-platform framework than for example the browser platform. Mobile devices such as smartphones usually have access to a camera which does not exist on the browser platform. Implementing the functionality of taking a picture with the camera in a cross-platform application might result in errors when using the browser platform.

# 3 Requirements

Before starting with the implementation itself it is crucial to know the requirements the application should meet and within what environment it will be used. Although a basic list of requirements was given at the start, some of these were not defined fully and were described more precisely at a later point.

After having talked about the basic amount of requirements with the supervisor of this project, a meeting of the KLIPS institute for the application was held. In form of a brain-storming session, future plans for the application were collected. However, the requirements collected in this meeting were very future-orientated and are not realistic as requirements for this first version of the application due to limitations in technology and the fact that the time for the implementation is limited.

Having collected all requirements from the meeting, a final list of requirements for the first version of the application was assembled. The result being the requirements defined below. These mainly consist of requirements collected in the brainstorm session but also requirements added by the project's supervisor for the purpose of simplifying future work on the application by other developers. Since most of the requirements collected in the meeting are very future-orientated, it is very likely that the application will be extended in the future.

The requirements can be separated into two different groups. These are functional and non-functional requirements. Both groups will be presented in the following.

## 3.1 Functional Requirements

Functional requirements define the quantity of functionalities the software application must meet. These describe the behaviour of the application in certain situations and the possibilities the user has with the application. In other words, functional requirements define what the user is able to achieve with the usage of the application. In the following sections, the functional requirements for the application will be described more detailed.

### 3.1.1 Cross-Platform: Browser, iOS, Android

This requirement was already mentioned in Section 2.2. It defines the software to be a cross-platform-application. Given that users should have easy and fast access to the functionality of the software system as described in Section 1.1, the decision was made to implement a mobile application. Mobile applications run on mobile devices such as smartphones and tablets.

Mobile devices mainly run on two different operating systems (OS). These are Android and iOS. iOS is Apple's OS for mobile devices, mainly used for Apple products such as iPad and iPhone. While Apple uses its OS mainly for their own products, Android is an open source software developed by Google. This property led to Android being used by other mobile device manufacturers, such as Samsung, Huawei or HTC. Since these are the OSs for mobile devices most widely spread, they were selected as platforms this application should run on. Because not all users might have a mobile device being able to run the software, the browser platform was added to the requirements additionally. This means that the application should also run inside browsers like Google Chrome or Mozilla Firefox.

Counting together, this makes the total number of three supported platforms: iOS, Android and Browser.

### 3.1.2 Basic Account-System

Before being able to add the actual functionality the application should provide, a basic account system needs to be implemented. The purpose of this system is the differentiation between users and to ensure that each user will only be presented his own data. Each user might have different interventions he is participating in. To be able to unlock the corresponding interventions for each user, the back-end of the system needs to be able to distinguish between different users. For this purpose, each account created should be globally unique.

For this system to work as intended, the account system has to provide the ability to create new accounts and login to already existing ones. To ensure that users are distinguishable, it has to be ensured that users only have access to their own accounts.

Considering that the back-end saves the provided account data and already provides the functionality of registration and login, the Patient-Module application needs to link client and server for this functionality. The back-end also defines the way this functionality can be accessed. For a successful registration, a HTTP request containing a globally unique email address, a globally unique account name and two entries of the same password is needed. To login to an existing account, the application needs to send a HTTP request containing the email and the password for the corresponding account. The application needs to provide the ability to send these request with all required data to the back-end.

In terms of data integrity and security described in Section 3.2 it is crucial that only authorized users have access to an account, implying that only authorized users can login to an existing account. Further, it has to be ensured that the user enters an email address that is actually his own considering that the application or the application's back-end contacts the user via email for the purpose of informing the user. After having created and confirmed an account, the user may login and use the actual functionalities provided by the application.

### 3.1.3 Account Administration

Account administration implies the functionality to change personal information. This excludes changing the email address for the sake of keeping the identity of the user. However, the user does not need to enter his personal information compellingly. Saving data like the age or sex should be optional. To be able to use the system, the user has to enter his email address and an account name. This data is required for the usage of the system.

### 3.1.4 Multilingualism

Considering that the users of the application will most likely be of international origin it is elementary to provide the ability to switch between different languages inside the application. For this purpose, the back-end already allows data to be transferred in more than one language. The user needs to have the ability to choose in which of the available languages he wants his data to be transferred. This data includes interventions, lessons and the question-elements of the lesson.

Furthermore, the user needs to not only choose the language of the data but also the language of the application itself. Given that the application will most likely be used for research purposes at Ulm University, the most used languages will most likely be German and English. Although most users might be fluent in the english and / or german language, it is likely that more languages will be added in the future to ease up the usage for people having trouble with these languages. This implies that adding new languages should be an easy task.

### 3.1.5 Authorization And Authentication

As already stated in Section 3.1.1, the account system is used for the back-end being able to distinguish the users it is communicating with. For this to work, the user needs to identify himself while communicating with the server. The back-end already offers a solution to this issue: JSON web tokens (JWT) [22].

On a successful server login, the client receives a JWT which provides the ability of authentication. The handling for this token has to be implemented on the client side, including automatic refreshing after the token expires, without any requests getting lost due to the expired token.

### 3.1.6 Offline-Usability

Users might want to download interventions and lessons they need to answer as they might be cut from internet connection at some point. Additionally, users might loose internet connection while trying to answer lessons. To prevent the loss of data and to ensure that users can always answer their lessons, the application should be able to run without internet connection.

To ensure that the user is able to work with the app as he intends, the relevant data needs to be stored directly on the device. This implies that the application has to synchronize the data between the back-end and the device self-sufficiently. This is crucial in terms of data integrity 3.2.1.

Adding to that, it is desirable that the application minimizes the amount of data that needs to be transferred between client and back-end. The client should only try to load data from the back-end if the data is not up-to-date.

To be able to use the application as it is designed, users need to be logged in to perform the desired actions. Hence, users should be able to log-in without having internet connection. Without an active internet connection, the application cannot communicate with the back-end, thus denying the ability to perform a back-end login. Consequentially, the application needs to manage its own account system which needs to be saved on the device to provide the ability to use all functionality of the application offline. By logging in without internet connection, the back-end will not be able to provide a valid JWT for the client. Hence, the application needs to execute a login request automatically after retrieving internet connection.

However, it is not advisable to store the login data of all users on every single device using the application due to security reasons. Also, to be able to use the application offline, the required data needs to be transferred to the device beforehand. Given that saving the data for all existing users would take a significant amount of storage, this is not an option. Therefore, only the data of the users that already logged into the app should be stored on the device. Consequential, the user has to log in one time with working internet connection to be able to use the offline feature of the application.

Although providing the main functionality of the application in offline-mode, there are functionalities the user will not be able to use in offline mode. These include submitting the answers to the server to finish a lesson, updating the profile data and password and synchronizing data with the server.

### 3.1.7 Displaying Interventions, Lessons And Answer-Sheets

To provide the user with enough information about the state of his therapy and to improve usability (3.2.2), interventions and lessons need to be presented visually to the user. This implies that the user has to be able to understand which lesson belongs to which intervention and implies that he can access them easily. Access in this case means that he can easily fill in the lessons and the visualization is self-explanatory. This also implies that the user has access to all answers he has already submitted. However, already submitted answers should not be changeable after submitting the answers to the back-end.

### 3.1.8 Filling in Lessons

To make the process of filling in lessons as easy as possible, a visualization of the lesson-elements needs to be implemented. Each lesson consists of a specific number of elements. These elements are grouped into blocks. Each block should be displayed on one "page", meaning the questions will appear beneath each other on the device's display. The user should be able to switch between the pages as he likes. The elements on each page usually build a structural unit. The separation of the elements into pages is already done by the CMS (1.2) and does not need to be addressed in the application. However, the application is responsible for displaying them correctly.

The lesson elements used can be divided into two groups: basic elements and questions elements. Basic elements do not ask the user for an answer but display additional information for the lessons instead. Question elements expect the user to put in an answer although answering question elements might be optional.

#### Basic Elements: Headlines, Texts And Media

At the current state, there are three types of basic elements.

1. **Headlines**
   These are simple headlines for a page or a component, introducing the next section of the lesson.
2. **Texts**
   These elements are simple texts to explain questions or to provide additional information to the user.
3. **Media**
   Media elements provide additional information in form of media. At the current state of system, there are three media-types:

   Images
   : Images can be of the type png, jpg or gif and display a picture or an image to the user.

Video
: Videos of short or long video records, supported with optional audio. Allowed formats at the current state are .ogg and .webm.

Audio
: Audio elements consist of an audio file. Supported formats are .ogg and .webm.

### Question-Elements

Besides basic elements, lessons consist of question elements. These elements ask the user to answer a question given by the question element. At the current state, there are seven types of these question elements, each of them asking a different type of question or expecting a different type of answer.

1. **Yes-No**
   This type of questions expect the user to answer the question with "yes" or "no"
2. **Multiple-Choice**
   This question-type offers a number of defined answer options and the user has to pick one or more as his final answer.
3. **Single-Choice**
   This type is similar to the multiple-choice, but only allowing exactly one answer.
4. **Slider**
   A slider gives a scale as possible answer. The scale has a defined minimum- and maximum value. It also defines the possible answer steps between the minimum and maximum value. The user answers this question by choosing a possible value on the scale.
5. **Short Text**
   This type of question expects a short text from the user as an answer.

6. **Long Text**
   Similar to the Short Text question-element, this question expects a text as an answer. While the Short Text element only accepts a few words, the answer to this question can be a lot longer. A popular use-case for this question-type is asking the user for feedback.

7. **Date**
   This question expects the user to answer with a date.

While some questions are obligatory, others may be optional. The visualization should inform the user if the question's answer is obligatory or optional. As soon as the user tries to submit his answers to the back-end, the application needs to check if all obligatory questions were answered. Furthermore, the application should only allow answers that match the corresponding question.

### 3.1.9 Quick-Saving Answers

Users might not want to answer lessons in one session. Since the user should not be forced to finish a lesson in one session, an ability to quick-save answers is required. This includes that the user is able to save his current state of answering a lesson to be able to continue the process at a later point. This implies that the user can quick-save all lessons he has to fill in and not just one.

Since this is only a comfort feature and the back-end does not provide any kind of functionality for this, quick-saved lessons will not synchronize between the different devices. If the user quick-saves a lesson, he will only be able to load these answers from the same device he has stored them on. All other devices will not save these answers and thus will not be able to load them.

### 3.1.10 Reminder

Users might want to be reminded if they have not submitted answers for their lessons yet. This version of the application aims on providing the ability of simple reminders for each lesson each week. These should tell the user which still have to be answered and should also be displayed while the application is closed.

Given that the browser module is limited in its functionality in comparison to mobile devices, this feature will not be supported on the browser platform.

## 3.2 Non-Functional Requirements

Non-functional requirements define requirements in terms of quality the application must meet. They allow the reviewer to judge a system in terms of quality, rather than describing features and behaviours of the application.

### 3.2.1 Security

Security describes if the system is behaving error-free while under external attacks or targeted disruption. Security can be divided into three different parts: confidentiality, integrity and availability, each described more precisely below.

#### Confidentiality

Confidentiality defines the protection from unauthorized access to data.

#### Integrity

Integrity defines the circumstance that data cannot be altered or destroyed accidentally or maliciously.

#### Availability

Availability defines the protection from data being made unavailable such as withholding data from users.

Considering that user data is stored in form of interventions, lessons, answers-sheets etc, as the application should run offline, it is needed to provide data security for the data stored on the device, implying the meeting of the three defined conditions.

### 3.2.2 Usability

Usability defines the ease of use and learnability of a software product. Usability can be interpreted as synonym for user-friendliness. One way of measuring usability is the time a user needs to achieve his goals by using the software. This requires fast understanding on how to use the system and how functionality can be accessed. Usability also takes into account how easy the usage of the system can be learned. It is desired that the application provides high usability through self-explanatory graphical user interfaces without the need to explain the user how to use it. This also includes the visualization of the application to be appealing to users.

### 3.2.3 Maintainability

Maintainability describes how easy the system can be maintained, meaning how easy it is to correct defects, maximize the application's lifetime, maximize efficiency and security, meet new requirements, add new functionality or make future maintenance easier.

### 3.2.4 Modularity

Modularity describes the separation of a system into different modules so they can be easily exchanged, altered or removed. Higher modularity means more and better encapsulated modules as this simplifies the actions described above.

Since this application will most likely be extended in the future, it is necessary that the application can be maintained easily and has high modularity.

# 4 Design

To ensure that the defined requirements can be fulfilled, a concept for the application structure is required. The choices made for this purpose are described below.

## 4.1 Choosing The Technology

The first decision to make was which technologies to use for the application, as they define the workflow of the implementation and the structure of the application. The fact that the application should be cross-platform for Android, iOS and browser determines possible technologies. Hence, the first challenge was to find a software framework for any programming language which is able to develop cross-platform applications for all desired platforms. After searching for eligible frameworks on the Web, the closer selection ended to be React-Native [6], Angular [14], Cordova [3] and Ionic [4] with Ionic already being based on Cordova and supporting Angular.

Ionic uses Cordova's functionalities and plug-ins by wrapping Cordova commands and extents the possibilities by adding its own functionality. In addition to that, Ionic supports the usage of the Angular Framework. Ionic thus offers the most functionality out of those three by using both other frameworks, leading to the actual competitors only being Ionic and React-Native.

In addition to Cordova's functionality, Ionic supports the usage of Angular and, coming with Angular, TypeScript. TypeScript is an addition to regular JavaScript and compiles to JavaScript while adding more functionality. React-Native also supports the usage of TypeScript, but does not support Angular. Ionic as well as React-Native encourage developers to reuse already written code as a result of source code working on all supported platforms. Using JavaScript and TypeScript makes it is easy for the developers to share and use their implemented code.

While both frameworks offer a good amount of possibilities and both seem to fit fairly well for the Patient-Module, it was Ionic's support of Angular what made the decision. Since both frameworks are fitting for the project, I chose Ionic as a result of my own preference. Angular itself is already a strong tool for cross-platform development for mobile and desktop applications. Considering the ability to learn both Ionic and Angular while using the Ionic Framework, I chose Ionic as technology for implementing the application.

## 4.2 Ionic

Ionic Framework is an open source toolkit for building mobile and desktop applications. It uses web technologies for building its' applications, these being mainly HTML, CSS and JavaScript. In addition to JavaScript, Ionic also offers the usage of TypeScript. Currently, Ionic has official integration with Angular [14].

The official Ionic website defines the goals of Ionic in the documentation [5] as follows. For the sake of reporting, the quotations are shortened.

> Cross-platform
> "Build and deploy apps that work across multiple platforms, such as native iOS, Android, desktop, and the web as a Progressive Web App [...]."
>
> Web Standards-based
> "Ionic Framework is built on top of reliable, standardized web technologies: HTML, CSS, and JavaScript [...]. Because of this, Ionic components have a stable API, and aren't at the whim of a single platform vendor."
>
> Beautiful Design
> "[...] Ionic Framework is designed to work and display beautifully out-of-the-box across all platforms. Start with pre-designed components, typography, interactive paradigms, and a [visually appealing] base theme."

> Simplicity
> "Ionic Framework is built with simplicity in mind, so that creating Ionic apps is enjoyable, easy to learn, and accessible to just about anyone with web development skills."

One very facilitating feature of Ionic is the given declaration of cross-platform HTML elements. Ionic uses different styling for each platform and each provided HTML element to create elements that fit into the styling of the platform. Thus, the developer does not have to create his own CSS styling for his elements but can use the elements offered by ionic. As an example, Ionic's input for dates will be compared to the one provided by HTML in the following. They both expect the user to enter a date. Both implementations do not use additional CSS styling.

Both elements use a so called "date picker" for the user input. This means that the user can open an additional window where a calendar is opened and he can easily pick the date he wants to enter. The HTML tag also offers the ability to enter the date by typing it while simultaneously making sure the user can only enter numbers in the correct format. Ionics element does not offer this since it is usually more comfortable to input the date with the usage of the date-picker, especially on mobile devices.

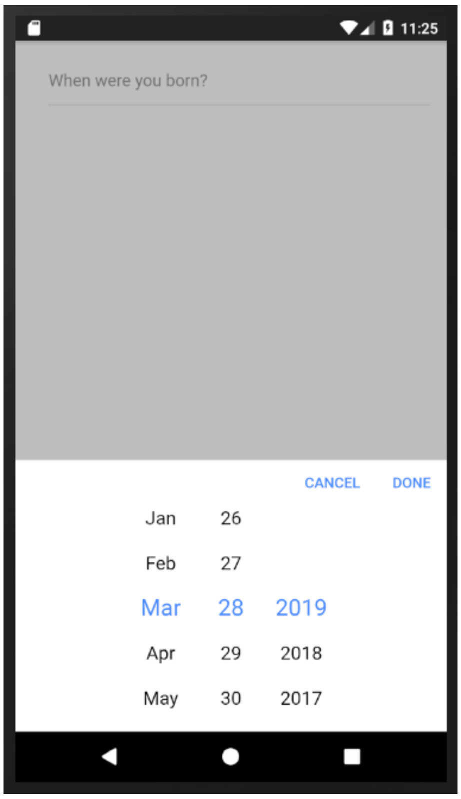
Figure 4.1: Ionic's HTML tags for date-time input

```
        <ion-item>
        <ion-label>When were you born?</ion-label>
        <ion-datetime></ion-datetime>
        </ion-item>
```

The implementation of a simple Ionic date input is presented in Figure 4.1. For the input to work as intended, Ionic suggests the wrapping of the element with an ion-item tag to build a structural unit and a label to inform the user about the data he should put in. This label will be shown to the user as part of the date input.

The result is shown in figure 4.2.

Figure 4.2: Ionic date input



By clicking on the input field, the screen will turn a bit darker, external actions will be disabled and the date picker will be shown. The user can now put in the desired value by choosing the corresponding day, month and year, presented by the date picker. As soon one of the buttons "done" or "cancel" is hit, the chosen value will be saved or dismissed and the date picker will disappear again. The same functionality as offered by the "cancel" button can be achieved by clicking on the screen above the date picker. Ionic also styles the input to match the current platform. For example, on the browser this input looks basically the same.

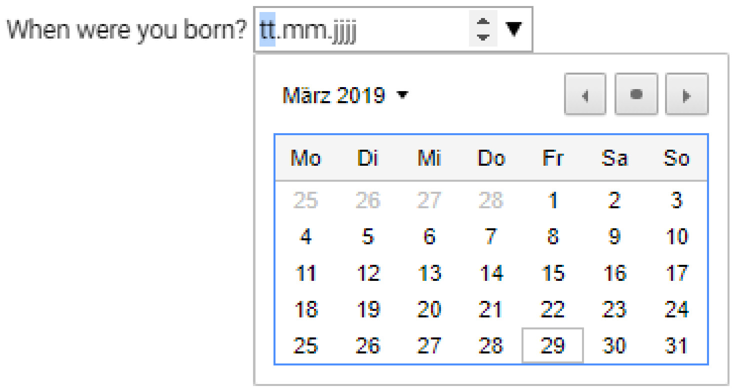
As Ionic also styles the HTML date input on mobile devices although not being an Ionic self-defined element, the HTML element presented below will be from the browser platform. HTML does not expect the date input to be grouped with a label into an item. However, the affiliation of the label to the input has to be defined by using and id for the input and setting the affiliation of the label to mentioned id. The corresponding source code is shown in Figure 4.3.

Figure 4.3: HTML tags for date-time input

```
<label for="birthday">When were you born?</label>
<input type="date" id="birthday">
```

This HTML code results in the view as shown in Figure 4.4.

Figure 4.4: HTML date input



| Mo | Di | Mi | Do | Fr | Sa | So |
|---|---|---|---|---|---|---|
| 25 | 26 | 27 | 28 | 1 | 2 | 3 |
| 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 11 | 12 | 13 | 14 | 15 | 16 | 17 |
| 18 | 19 | 20 | 21 | 22 | 23 | 24 |
| 25 | 26 | 27 | 28 | 29 | 30 | 31 |

In contrast to the simple HTML element, Ionic's element is already styled with CSS without the need for the developer to add any CSS classes or additional styles, whereas the HTML element looks pretty simplistic and is not responsive, Ionic's element fulfils these tasks without any further addition to the element. The look is much more user-friendly and more appealing and is responsive to the screen size.

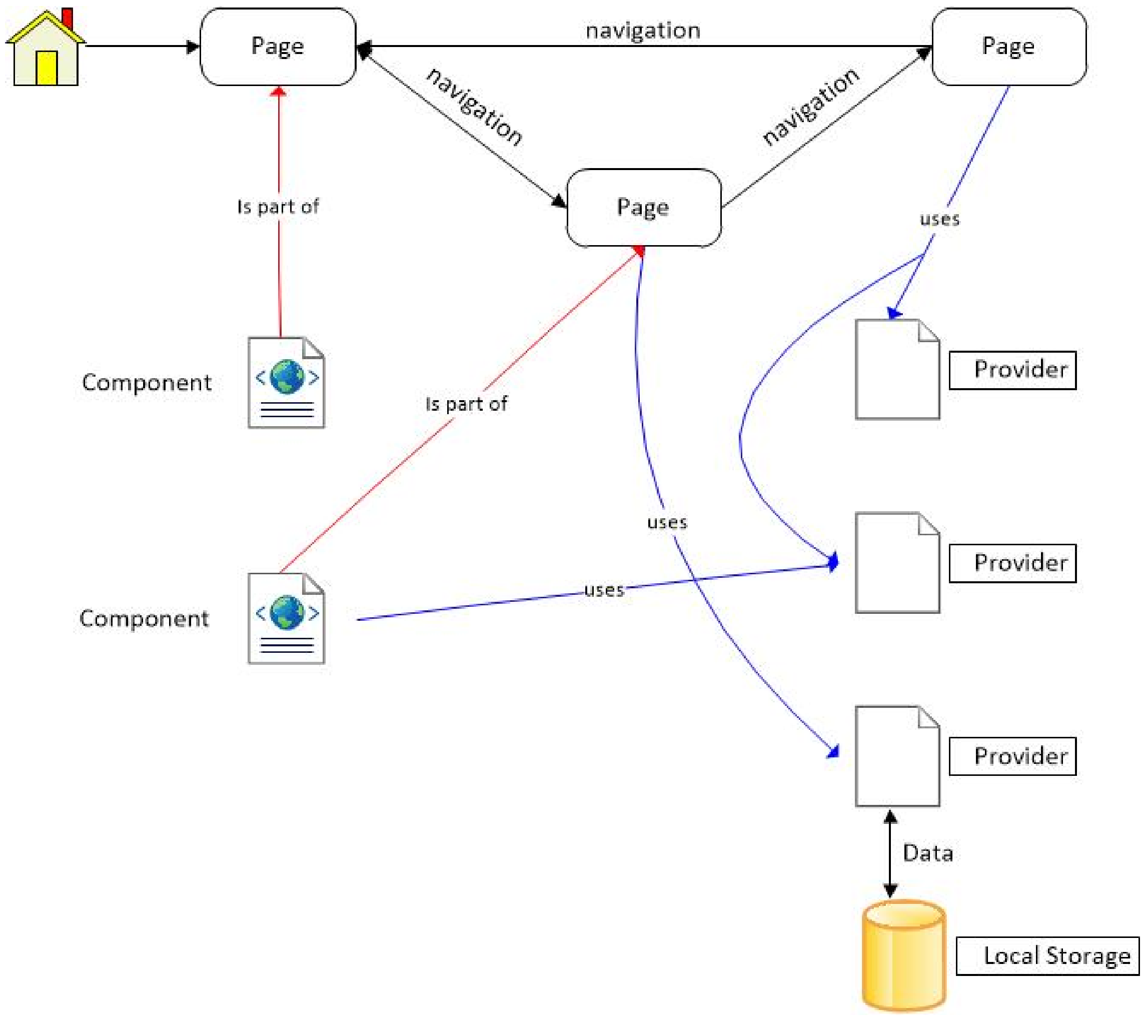
Ionic also offers a very modular and easy understandable application structure, making it easy to implement, alter, switch out or delete small parts of the application. Furthermore, Ionic provides a large scale of usable plug-ins, including most of Cordova's available plug-ins.

The usage of Angular additionally extends the possibilities of Ionic through finished and usable Angular modules such as the HttpClient. However, this work does not want to give a presentation for the Ionic framework as a whole. Other features can be explored in Ionic's documentation which can be found in [4].

## 4.3 Structure

As described before, Ionic provides a modular and easy understandable architecture for its applications. For developing, Ionic differentiates different modules such as Components, Providers and Interfaces. However, not all modules will be presented in this thesis.

Figure 4.5: Software Architecture



The basic architecture of the software is shown in Figure 4.5. It mainly consists of three different module types: The Pages, the Components and the Providers. Pages present the Graphical User Interface (GUI) to the user and provide the ability to show information and access the application's functionality. Components basically fulfil the same purpose, although they do only implement smaller GUI elements with a given functionality which can then be used by the Pages. Providers are used to provide all functionality for the application that is not linked to a user interface, such as handling server requests or handling the local storage. These modules are not accessed by the user directly but are called by the Pages and Components. The pages used for this application are described in Section 5.2. Each module will be defined more precisely down below.

### 4.3.1 Components

Components provide a possibility to create GUI elements and bind them to a controller which handles the functionality behind them. Therefore, components usually consist of three files. One HTML file, one SCSS file and one TypeScript file.

The HTML defines the elements of the component's GUI. Everything the user can see and interact with on a components GUI is defined in this file. There is already a good amount of explanations and tutorials on how to build GUIs with HTML on the web so this work will not cover this topic in depth. In addition to plain HTML, Ionic offers a variety of different additional HTML elements defined by Ionic as already mentioned in Section 4.2.

The SCSS file is responsible for styling the GUI by making the HTML structure defined in the HTML file visually more appealing. It is used to style the elements given through the HTML definitions. SCSS uses a more simplified and developer-friendly syntax for CSS code. HTML and SCSS together allow the developer to build a GUI and apply visual designs to it, thus providing the ability to create appealing interfaces for the interaction of human and application. Again, this work will not cover CSS and SCSS in depth.

The TypeScript file embodies the controller for the GUI. Its function is to handle all functionalities the component should provide which is accessed by the user through the GUI. These might include opening new pages, loading data from storage or storing data into the storage. In addition to the human-application interaction, the controller handles all background functionality of the application the user cannot see by using providers (4.3.3).

To extend the modularity of the application, following components were defined. This short overview only covers the most important ones.

- Language-Setter
  This component is responsible for the translation of all needed text output provided by the GUI. It does not provide own GUI elements but only offers the ability of translation. This implies that it is not visible for the user.
- Nav-Bar
  The function of this component is to provide a possibility for the user to navigate through the app by changing pages. It contains the Navigation Menu, which will be introduced in Section 5.2.2, and a logout ability.
- Intervention
  This components contains the data of an intervention as provided by the back-end and is able to display this data with the help of GUI.
- Lesson
  These components basically have the same task as Intervention components, though they contain lesson data instead of intervention data.
- Lesson Elements
  Each lesson element has its own component, consisting of GUI elements styled with SCSS and a controller. However, not all components for these lesson elements will be described in detail as they only differ in the provided GUI elements and the data they contain. The functionality for these elements is generally the same. They provide the ability to display the element's information, such as the included question or answer options, and answer question elements.

These components are used by the pages to enable the user's access to their functionality. Some of them use providers to implement additional functionality. For example, all pages use the Language-Setter component to enable translation for the pages.

### 4.3.2 Pages

Pages define the actual 'pages' in terms of the GUI between which the user can switch inside the application. They can be compared to different web-pages on a website. The application defines a root-page which will be the page displayed to the user when he starts the application. Usually, other pages can be accessed through a navigation, this being either a group of links or buttons the user can click on, different tabs or any other kind of structure that allows the user to access different pages or any other kind of functionality. In Ionic's context, each page acts as an entire view.

Angular itself does not have pages. It uses its components as pages. In contrast to Angular, Ionic differentiates between pages and components. The structure of both is very similar though with pages defining a special kind of component. As every component, pages consist of a HTML, a SCSS and a TypeScript file while each file describes the same functionality as already described in Section 4.3.1.

Ionic interprets each page as a stand-alone view, meaning that each page is one complete GUI. This does not mean that all HTML and CSS code has to be defined in the page's HTML and CSS files though. Pages can use components to provide additional GUI and corresponding controlling inside the page as shown in Figure 4.5.

### 4.3.3 Providers

Providers are Angular's attempt on providing any kind of service. They are used to provide basic functionality that is not bound to pages or components. They are used to group functionality by the type of functionality they provide. Therefore, common usages of providers are handling server requests or local data administration.

In Ionic, these providers consist of only one TypeScript file where the functionality of the provider is implemented. The providers can be used by pages (see Section 4.3.2), components (see Section 4.3.1) or other providers to access their functionality.

To provide an example, the most important providers used by the application are listed below.

- ServerRequestProvider
  This Provider is responsible for any kind of client-back-end communication. It defines a collection of methods sending requests to the back-end, using Angular's HttpClient module. The methods then return the response of the back-end. This provider is mainly used by other providers needing back-end communication, such as the SInterventionHandlerProvider or the LessonHandlerProvider.
- LocalStorageProvider
  For the administration of the local storage, the LocalStorageProvider is used. This module offers methods for storing, reading and deleting any kind of data for the local storage.
- InterventionHandlerProvider
  The InterventionHandlerProvider contains all functionality concerning the administration of interventions. It uses the ServerRequestProvider to exchange intervention data with the back-end and uses the LocalStorageProvider to administrate the local saving of interventions.
- LessonHandlerProider
  This Provider basically implements the same functionality as the InterventionHandlerProvider, but covering the functionlities needed for lessons.
- RequestMiddlewareProvider
  This Provider used indirectly by any Page, Component or Provider sending requests to the back-end. Indirectly means that this provider is not called by other modules but implements Angular's HttpInterceptor module to adjust all requests the application sends to back-end. It is mainly used for token refreshing (3.1.5).

## 4.4 Local Storage

Ionic offers the usage of SQLite as a local database. Since the application must run on the browser platform which is no able to use SQLite due to its limitations, SQLite was not used for this application. However, Ionic provides another possibility to store data locally. The so called "local storage" can save key-value pairs inside the storage of the application.

However, the local storage only has the ability to save data as text. Given that all user data including interventions and lesson should be saved on the device, it is necessary to translate data objects as used by the applications to text and vice versa. This feature is already provided by the Java Script Object Notation (JSON) which is also included in Ionic. Combining JSON and Ionic's local storage, the ability to save and load data to the devices storage is provided even on the browser platform.

However, Ionic's local storage is only acceptable as storage for data that is not security sensitive due to its very poor security standards. Thus, either a different storage or an encryption tool has to be used to ensure data security and integrity.

Finally, a different storage was chosen for security-sensitive data and Ionic's local storage was chosen for not security sensitive data. The storage chosen for security-sensitive data is a plug-in for Ionic named "Secure Storage" and will be covered in Section (5.4.1).

# 5 Implementation

After collecting requirements and constructing a design, the actual implementation for the application was done. This chapter addresses the implementation process and its results. This includes an introduction to the visualization, the approach of the implementation, a description of the challenges faced while implementing as well as a small introduction to used plug-ins.

## 5.1 Approach

To ensure that the application will work as intended and all requirements are met, the process of implementation needs to be planned beforehand. For this purpose, different methods were used. The most important ones are described in this section.

### 5.1.1 Defining Requirements

Before being able to start the implementation it is needed to define the requirements for the application as already described in Chapter 3. Considering this circumstance, the first action was to collect all requirements for the application. The process for this first step is described more precisely in the referenced chapter.

### 5.1.2 Issue Tracking

Issue tracking is a common way of keeping the outline the features already implemented and those which still have to be done. An issue tracker provides a detailed overview over all tasks that have to be worked on. To work as intended, these tasks should be as detailed and small as possible and should be completable in a preferably short amount of time. These so called issues are not all defined at the start of a project but can be added throughout the whole software-engineering process. They include bugs found while testing the application and additional work that occurs at a later point.

While implementing the required features of a software system, it is most likely that developers cannot remember all required work without taking notes. This defines the main reason for issue tracking. Furthermore, other team members can keep track of what work still needs to be done. Existing tools that support issue tracking are Pivotal Tracker or GitLab's issue feature. These tools are mostly used by teams that develop software projects together to keep track of the work done by others and what is not yet finished.

Although not that formal, this concept was used for the implementation process of this application. Starting with the project, all basic requirements were noted in form of issues into the provided GitLab repository for the source code. As the implementation continued, issue tracking tool was switched out as a result of the group size being only one person.

Given that the amount of time needed to write down the issues as formal as GitHub intends is high compared to a local text-log, GitLab's issue tracker was switched out by a simple text log. Thus, the issue tracking continued in form of a text file where all issues were logged in form of a small explanations.

### 5.1.3 Weekly Goals - Personalized SCRUM

To provide a structured implementation process the concept of SCRUM was chosen. SCRUM is an agile method of software developing, meaning that the process of implementation and the implementation itself is not planned completely from the start but is planned for each part separately and right before this part is implemented. This offers high flexibility throughout the whole implementation and the possibility to adapt to problems or changes from the outside.

The basic operating principle is as follows. The implementation is divided into a number of so called sprints which are defined by a period of time. The lengths of these sprints are usually the same. Before starting a sprint, the developer team plans what will be done during the sprint. This is usually defined through issues as mentioned in Section 5.1.2. These tasks, or issues if issue tracking is used, are distributed to the team members while taking in count what each member can possibly achieve during this time. At the end of the sprint, the team compares the outcome with what was planned and, if necessary, writes new issues. Afterwards, the next sprint is planned and the issues to be fulfilled are distributed again.

As a reason of the group size being only one person, this principle was personalized to fit for this developing process. While the basic idea remained the same, the sprint length was not the same for all sprints due to personal reasons. Also, no sprint meetings were held since there was simply nobody to talk to and issues did not have to be distributed. Considering that the developer team consists of only one person, the usual distribution of SCRUM roles such as SCRUM master was redundant.

To provide an overview over the process the implementation went through, the following will describe the single topics implemented. Considering that each topic might be too much work for one week, these topics were internally partitioned into issues (5.1.2) with each sprint covering a specific amount of these issues.

1. Login
   Considering that most functionality of the application needs user authorization, the first part to implement was a functional login. Without this being implemented it is hard to test implemented functionality which needs authorization. This topic included the issue of setting up functional requests to the back-end.

2. Intervention Overview
   After a successful login, the user should be able to access his data, this being participating interventions, lessons and other. To actually be able to use the application's functionality, it is essential to be able to access this data with the help of graphical user interfaces, leading to the Intervention Overview being the second part to implement, including loading needed data from the server.

3. Token-Refreshing
   As the login already implied the basic usage of the provided JWTs from the server, this feature had to be extended to provide the ability to refresh expired tokens automatically. Since this is a feature used by the whole application, it was helpful to implement this feature as early as possible.

4. Lesson Overview
   After being able to handle JWTs and load data from the server, the goal was to be able to actually view lesson content in form of questionnaires. For this to work, lessons containing their elements had to be loaded from the server to provide a GUI for the user to be able to start the mentioned questionnaires.

5. Multilingualism
   Taking into account that all texts used inside the GUI should most likely be translated, it was important to implement multilingualism as early as possible to prevent the need to switch out all texts with the translation functionality. This might result in great effort if much text has to be switched out.

6. Viewing Lesson Elements
   As answering lessons is the central functionality of this application, the lesson elements needed to be presented visually to the user. Considering that the amount of different possible elements is relatively high and a GUI element for each lesson element had to be created, this was possibly the part which took the most time to implement, thus being split to more than one or two sprints.

7. Answering Lessons
   The next topic to be implemented after being able to view lessons with their elements was actually answering the presented lessons. Before submitting answers to the back-end, the quick save functionality was implemented in terms of more possibilities in testing the implemented features.

8. Local Storage
   Given the basic features of the application being implemented, the next step was to provide the ability to use the implemented functionalities while the application was cut from internet connection. This implied saving the already loaded data as well as preparing the application's local storage for future extensions in terms of functionality.

9. Offline-Usability
   As the data could now be saved locally inside the local storage, is was desired to make the application actually usable if internet connection was cut. To achieve this functionality, the application has to detect if a connection is present and use the local storage instead of trying to load data from the server if not. This also implied the implementation of a local account system so the user can login without the application's need to communicate with the back-end since local saving of the data is irrelevant if the user cannot login if offline.

10. Personal Data
    After finishing the central functionality of offline usability and answering lessons, the next step was to realize the handling of personal information. Considering that future plans for this application include some kind of social network, it is desired to be able to set personal information such as name and sex to be able to identify other users if wanted.

11. Registration
    Since this functionality had no high priority, it was implemented relatively late in the implementation process.

12. Lesson Media Elements
    These lesson elements were not implemented from the start as it took a great amount of effort find a solution for saving these elements locally. Hence, they were pushed to the end of the implementation.

13. Notifications
    For the priority for this feature not being high and just as media elements taking great effort, this topic was the last to be implemented.

### 5.1.4 Code Refactoring

As a result of not having worked with either TypeScript, Angular or Ionic before, the knowledge about the given technology increased with adding new functionalities and working on issues. Although having already done research, tutorials and small test implementations as exercise beforehand, it is undeniable that the knowledge of the used technology was low compared to experienced Ionic, Angular or TypeScript developers.

Hence, the understanding of the technology increased as the implementation of the application progressed. Given a higher understanding, already implemented source code appeared ineloquent and sometimes obsolete. Considering that refactoring usually results in more effective and more efficient source code, this process usually was executed in form of adding new issues when new discoveries concerning the technology were made or when the understanding increased as a result of mastering challenges or facing problems.

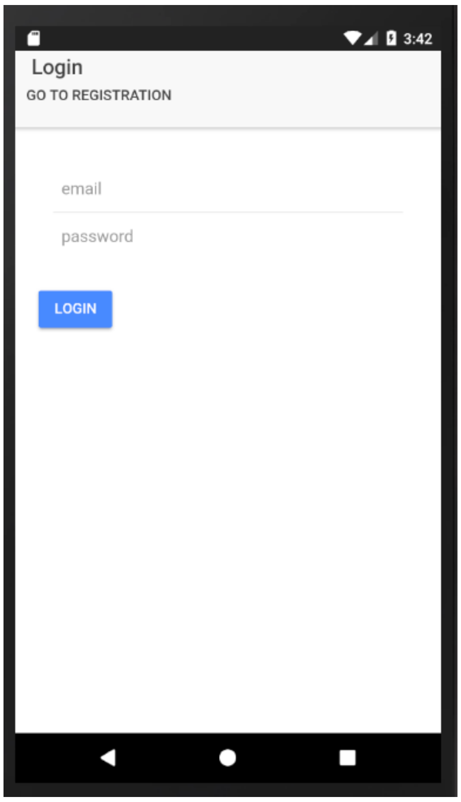
The process of refactoring covered as much parts of the application as possible to keep a structured and consistent code.

## 5.2 User Interface And Navigation

The finished application can be divided into the different pages and views the user will be able to access. This section provides a small introduction to the visual appearance of the pages and how to use their functionality.

### 5.2.1 Login / Registration

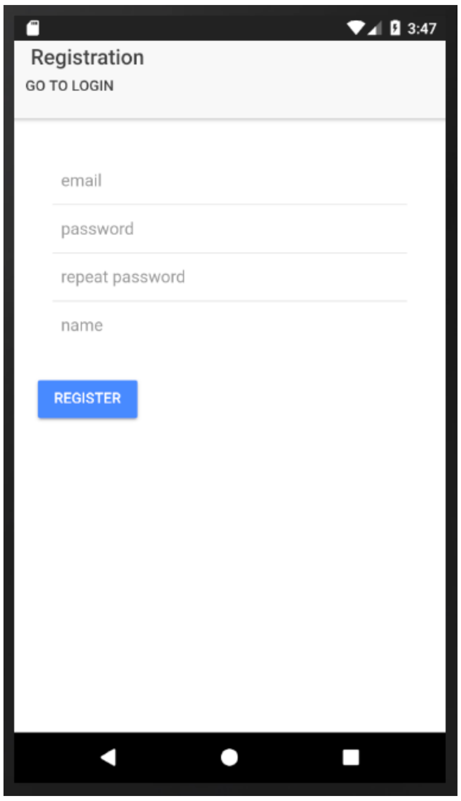
Figure 5.1: Login Page



The initial page displayed to the user on application start-up is the login page shown in figure 5.1. Before being able to use the elementary features of the application, the user needs to log in. For this process, the user's email address is needed to identify the user and a password has to be entered to prove the user's authorization. For this process the user needs to fill in the two text inputs labelled with email and password. To conclude the login process, the user needs to hit the "Login" button below the input fields.

Finishing this process by having entered correct login data and pressing the "Login" button will lead the user to the "Home Page" described in Section 5.2.3. If the user enters incorrect data, an error window will pop up and inform the user about the failed login. Assuming that the user does not have an account yet and wants to create one, the button "Go to registration" underneath the Login title can be pressed. This will result in the view changing to the registration page as shown in Figure 5.2.

Figure 5.2: Registration Page



Instead of providing the ability to log into an account, this view’s functionality is the creation of accounts. For creating an account, the user needs to fill in four inputs: his email address as an identifier for the user, a password to be used as authorization security, a repetition of his password to ensure that the user enters the password correctly and an account name. Both the email as well as the account name are globally unique. Consequentially, the registration process would fail if a different user already created an account with the entered email or account name, resulting in an error window pop-up to inform the user. Additionally, the inputs must meet different specifications to be accepted. These include that the entered email has the format of a legit email address, both entered passwords have to be identical and the password length is at least 8 characters. Not meeting any of those requirements will result in an error window and the need to repeat the registration process.

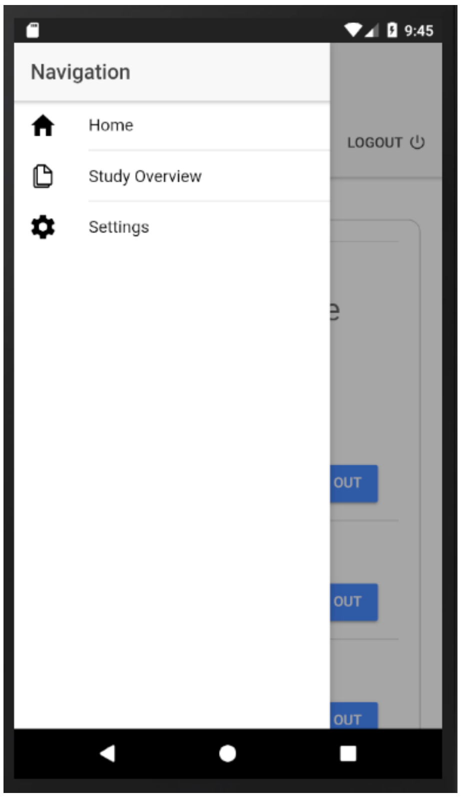
After a successful registration, an email will be sent to the user containing a link which will activate the user's account. The user cannot log into his account until this confirmation link is clicked and thus the account was activated.

For data integrity reasons, the registration will only work if the user has active internet connection.

### 5.2.2 Navigation Menu

The navigation menu is placed on the top left corner of the app, the visual appearance is shown in Figure 5.3 and can be found on the Home Page (5.2.3), the Intervention Overview Page (5.2.4) and the Settings Page (5.2.7). It is accessed through a button which will open the menu.

Figure 5.3: Navigation Menu

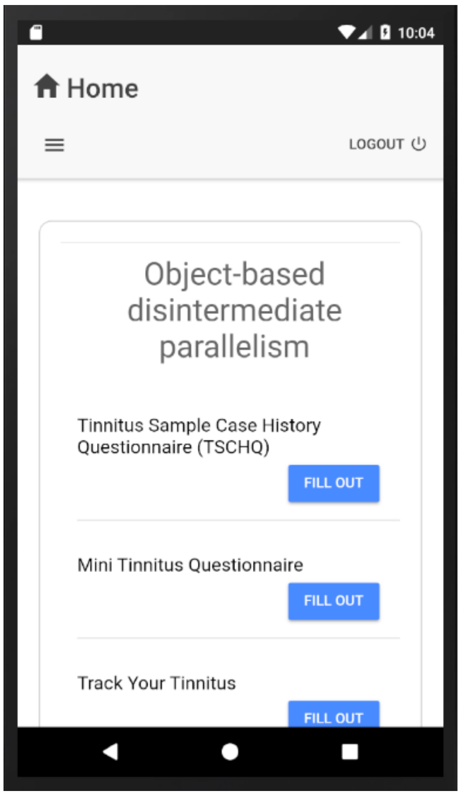


The menu can be used to navigate through the application, meaning that the user can control which page will be displayed. The navigation menu currently contains three different pages: the Home Page (5.2.3), the Intervention Overview (5.2.4) and the Settings Page (5.2.7). Other pages might be added with application extensions. All of these pages have access to the navigation menu as well as providing the ability to log out of the application. For this purpose, a logout button is located in the top right corner of the application.

### 5.2.3 Home Page

The Home Page is the starting view the user will see after a successful login (5.4).

Figure 5.4: Home Page

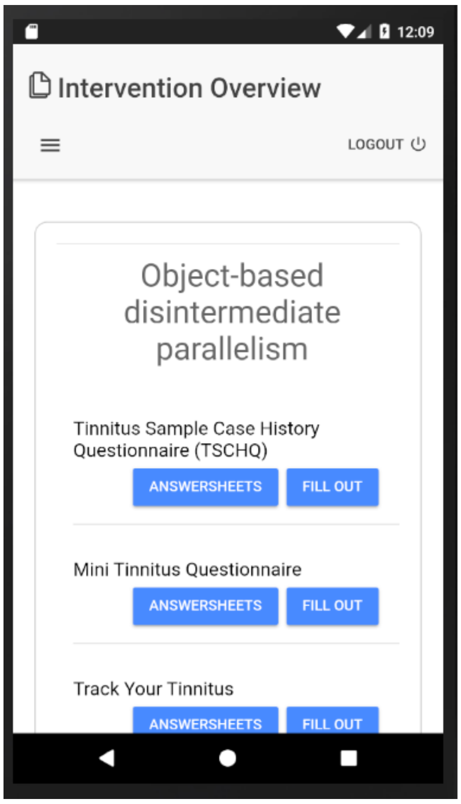

10:04
Home
LOGOUT
Object-based disintermediate parallelism
Tinnitus Sample Case History Questionnaire (TSCHQ)
FILL OUT
Mini Tinnitus Questionnaire
FILL OUT
Track Your Tinnitus
FILL OUT

The page provides an overview of interventions and lessons the user is participating in. On the home page, these are filtered and only those which are currently active are shown to the user. Active in this case means that the user has not finished them yet and the current day and time fits the time period they can be answered in. For each intervention, a list of unfinished lessons is displayed. The answer process of these lessons can be started by clicking on the "Fill out" button offered by each lesson.

### 5.2.4 Intervention Overview

The structure of this page is very similar to the structure of the Home Page (5.2.3). The central functionality of this page is to show the user the interventions and the lessons he is participating in as shown in Figure 5.5. The difference being that the home page filters all interventions and lessons as described in 5.2.3.

Figure 5.5: Intervention Overview Page


Intervention Overview

Object-based disintermediate parallelism

Tinnitus Sample Case History Questionnaire (TSCHQ)

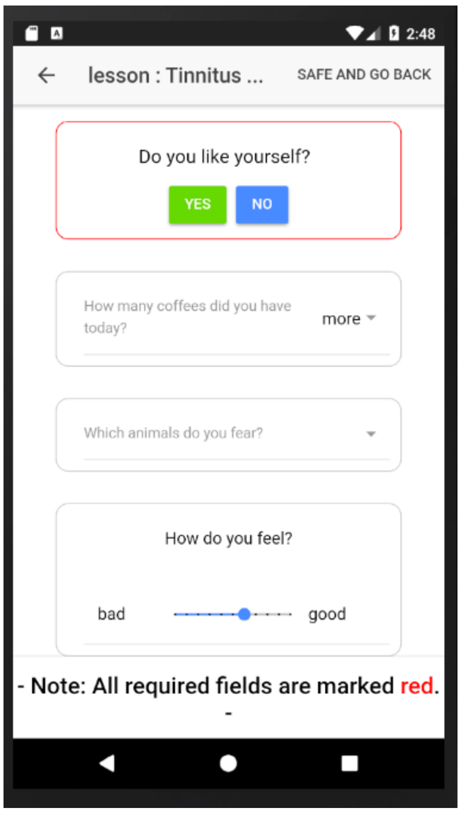
Mini Tinnitus Questionnaire

Track Your Tinnitus

The intervention overview will not filter interventions and lessons but will mark them with a corresponding color representing their current status. Currently, the application will mark the finished ones grey while active ones are marked black. This might get extended with more features added in the future. Each lesson of an intervention provides two possible user actions. The first one being the possibility to start answering the lesson. This will lead to the start of the process described in 5.2.5. The second action is accessed through the button "Answersheets". This action will lead the user to the answer-sheet overview (5.2.6) for the corresponding lesson.

### 5.2.5 Filling In Lessons

Figure 5.6: Filling in lessons


2:48
lesson : Tinnitus ...
SAFE AND GO BACK
Do you like yourself?
YES
NO
How many coffees did you have today?
more
Which animals do you fear?
How do you feel?
bad
good
- Note: All required fields are marked red. -

As soon as the user starts the process of filling in a lesson, he will be shown an overview of the lesson he is going to fill in. This usually includes a title and a description and sometimes some additional information about the lesson. This data is provided by the back-end and is not part of this application. This overview includes a button which will actually start the lesson by showing the lesson page presented in Figure 5.6.

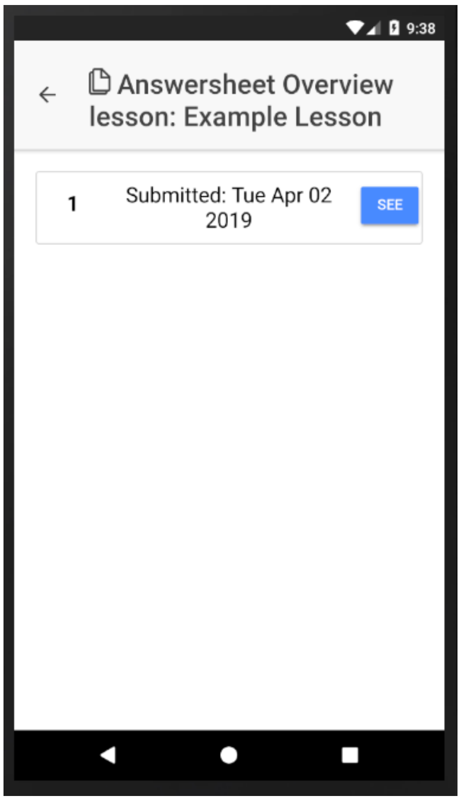
This page will show the elements defined for the lesson (see 3.1.8) grouped by pages and in the correct order. A page in this case is a structural unit which starts at the top of the display and ends at its bottom. If a page contains more elements than the display can show, the user can scroll down to see the missing elements.

If the device stores quick-saved answers (see 3.1.9), these will be loaded for the corresponding elements.

Pages can be swapped by either swiping to the right or left or with the "previous page" and "next page" buttons at the end of each page. If the user reaches the end of the lesson, the "next page" button will be replaced by a "finish lesson" button. If this button is clicked, the application will check if all requirements for the lesson are met. If this is the case, the application tries to submit the answers to the back-end. Considering that the request is successful, a new answer-sheet will be created locally and the intervention overview page will be shown to the user. If not, an error window will pop up and inform the user about the unsuccessful finish. Since this action needs the application to communicate with the back-end, an active internet connection is required.

### 5.2.6 Answer-Sheet Overview

Figure 5.7: Answersheet Overview

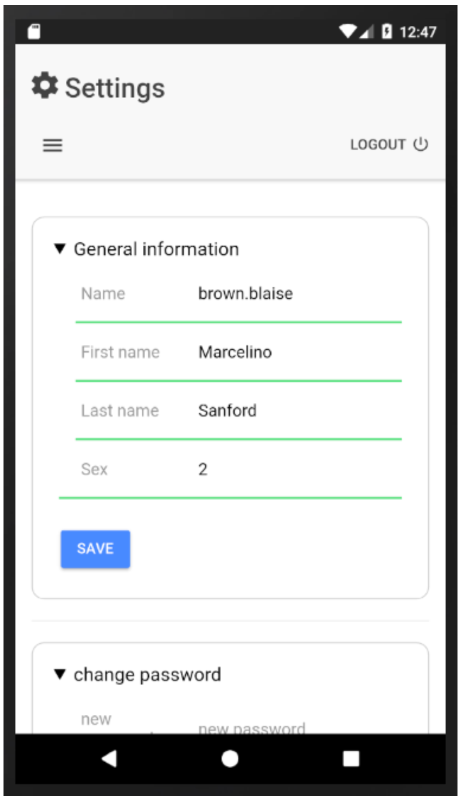


The purpose of this page is to show the user all answers for a given lesson he has already submitted. Whenever the user finishes a lesson and thus his answers have been submitted to the back-end, an answer-sheet is created both on the back-end and on the device. These answer-sheets will be shown on the answer-sheet overview page with the date and time the answer-sheet was submitted at as shown in Figure 5.7. This page exists for each lesson the user is participating and shows all created answer-sheets for this lesson.

The user has the ability to open these sheets. This results in the user getting displayed the same view used for filling in lessons (5.6). However, using this feature, the user will not be able to change answers of any element since the purpose of this feature is to only show the user which answers were submitted.

### 5.2.7 Settings

Figure 5.8: Settings Page



▼ General information

| | |
|---|---|
| Name | brown.blaise |
| First name | Marcelino |
| Last name | Sanford |
| Sex | 2 |

▼ change password

| | |
|---|---|
| new | new password |

This page allows the user to check and to change his personal information. This includes his first name, last name, his account name and his sex. The user can easily change this data by changing the entries in the input fields and hitting the save button after. The Settings Page is shown in Figure 5.8.

This page also handles the user's preferred language. This is done by changing the entries of the application language field and the lesson language field. The first is responsible for the language the application runs in. Changing this entry will result in changes for all buttons, headlines and labels the application uses. Changing the second entry will lead to the the data of interventions and lessons received from the back-end being in the chosen language.

Additionally, the user can change his password used for his account on this page. For this purpose, three input fields are provided. The first one checks the user's current password so the changes can only be made if the user is able to authorize himself. The other two expect the new password while both inputs follow the same rules as on the registration page (5.2.1). If all is entered correctly, the password will be changed on the back-end and on the local device.

Changing any of the data except the language requires an active internet connection.

## 5.3 Challenges And Problems

In the process of implementation it is usual that problems appear and challenges have to be faced. This Section covers the problems and challenges that appeared while implementing this application.

### 5.3.1 Defining Modules For The Application

The very first challenge faced was the actual navigation of Ionic. Ionic's navigation is conceived like stacks known from computer science. A stack stores values by putting them on top of each other and taking them off again. Storing a value will result in another stack element being placed on top of the stack ("push" operation) and reading a value will remove the top element of the stack ("pop" operation).

Figure 5.9: Source Code for navigation in Ionic

```
export class HomePage {
  constructor(private navCtrl: NavController) {}

  public pushSettingsPage() {
    this.navCtrl.push(SettingsPage);
  }

  public setRootSettingsPage() {
    this.navCtrl.setRoot(SettingsPage);
  }
}
```

Ionic's stack navigation works accordingly. It defines a root-element for the stack on which different pages (see 4.3.2) will be pushed and popped. The display will always show the top page element of the stack. Each pushed page will receive a button in the top left corner with an arrow icon that, if pressed, pops the current page from the stack and the page below will be shown again. Additionally to pushing and popping, Ionic's navigation offers the ability to set new roots. This will empty the current stack and set the new page as root-page for the stack. Thus, navigation is achieved through the orders in the source code below 5.9.

However, working with tutorials from the start, the navigation given through these tutorials followed a slightly different pattern. Instead of calling page modules directly, quotations were used. Instead of *push(SettingsPage)*, *push("SettingsPage")* was used. This is a commonly used way of routing, but requires the page-to-push to be defined as lazy-loading page. Since lazy loading was not used for these pages, this led to an error which was thrown when trying to access a page with the push operation in specific cases. This was an error occurring at the very beginning of the implementation process, for what reason the knowledge about Ionic was limited and checking internet forums for the solution to this error seemed like the most reasonable option.

The solution found on the internet to this exception thrown by the compiler as consequence of this issue was defining the page module the navigation should push to differently in the applications module file. Although solving the navigation issue, the provided solution was not acceptable as it resulted in not being able to use self-defined components on this page anymore. After taking a good amount of time finding the actual issue, it was solved by simply not using the quotations. This is not the type of issue one would expect to occur, regardless taking a lot of time to be fixed.

### 5.3.2 Back-end Requests

Due to the application's need to exchange data with the back-end, it is needed to implement some possibility for the application to communicate over the internet. The back-end already offers a a way for this purpose. It is designed to respond to incoming HTTP requests if the user sending the request is authorized. Thus, the application needs to implement this form of communication.

Angular's HttpClient does exactly that. The module can send different types of requests to a given URL and receive the response. For transmitting data, it uses JSON as data structure. Data retrieved from the HTTP response will also be transmitted as JSON if present.

Taking into account that calling HTTP requests is an essential part of functionality of the application, a small code example for a simple HTTP post request is provided in Figure 5.10.

Figure 5.10: Example Request for back-end communication

```
export class RequestProvider {
  constructor(private http: HttpClient)

  public submitData(data, url) {
    this.http.post(url, JSON.stringify(data),
      {observe: 'response'}).then((response) =>
      if (response instanceof HttpResponse) {
        localStorage.set("data",
        JSON.stringify(response.body);
      }
    )};
    doSomething();
  }
}
```

The provided example will send a post request to a given URL while adding given data to the requests body. Angular's HttpClient uses asynchronous requests, recognizable by the 'then' keyword. This implies that the application will call the request with the ***this.http.post*** method and will immediately execute code below the request call, in this case the ***doSomething*** method. All code declared inside the 'then' keyword will be executed as soon as the response is received, the response included in the 'response' wildcard. In this example, the received data will be stored inside the local storage. The option "observe: 'response'" will lead to the HttpClient providing additional information about the response from the called server such as the type of response. Since requests usually take some time to receive an answer, ***doSomething*** will be executed before the code inside the 'then' expression.

Since the local storage 4.4 and the secure storage 5.4.1 both use simple text stored values, the transmitted JSON data can easily be translated to a simple text and stored inside the storage.

### 5.3.3 Authorization And Authentication - JSON Web Tokens

The handling of JWTs is needed for authentication since the back-end already offers this as solution for authorization. The following will describe the way these token are used and handled for this purpose.

After a successful login of the user at the back-end, the back-end creates a JWT and attaches it to the client's response. The client can now use this JWT to authenticate himself when sending another request to the back-end by attaching the JWT to the request. The back-end itself can determine which user is sending the request by reading the JWT. To use the authentication method offered by the back-end the corresponding token handling on client side has to be implemented. The JWT needs to be saved on the client and attached to each request which requires user authentication. Except for login and registration, all requests need user authentication. Additional, the JWT expires after some time as a result of security guidelines. After the JWT expires, all requests sent to the back-end with the expired token will be responded with an HTTP-Error-Response by the back-end. Logging the user out of the application at this point is no reasonable option as this might lead to a loss of unsaved data for the user. To address this issue, the application needs to detect the expiration of the JWT and update it if necessary. The functionality of refreshing the JWT is already implemented on the back-end. Sending the corresponding request allows the client to refresh the JWT if the expired JWT is attached to the mentioned request without the need to perform another login.

To implement this functionality, a middleware was implemented using Angular's HTTP interceptor. This tool allows to intercept any request sent from the application. Thus giving the ability to detect expired tokens and refresh them if necessary. This is implemented as follows: Before actually calling the request, the interceptor checks if a token refresh is currently in progress which is recorded in form of a boolean. If so, the request simply gets queued. If not, the interceptor adds the current JWT to the request and sends it to the back-end. After this, the interceptor observes the back-end's response. If the token is expired, the back-end will respond with a 403 HTTP-Error-Response containing the message "Token has expired.". The interceptor catches this error and immediately sets the current refresh boolean to true so following requests will be queued. Continuing, the application sends the refresh request with the current, expired token.

As soon as the response containing the new, valid token is received, the token is saved on the client, the refresh boolean is set back to false so following requests are not queued any more and queued requests are notified. This results in the queued requests being adapted to the new token and then being sent.

### 5.3.4 Local Saving Of Data

Defined by requirements, the application should be able to use a storage for account information and user depending data. The challenge is to find a way to store data on each platform. As a result of having already worked with databases, the first idea for the storage was the usage of a database, such as MySQL, SQLite or any other possible database. Although SQLite is supported by Ionic, it does not work on the browser platform due to the platform's limitations.

Searching for alternatives working on the browser, Ionic's local storage was detected. This storage works on devices as well as on the browser. However, it is very insecure in terms of data security. It is not hidden and does not encrypt the data stored. To face this problem, Ionic also provides an alternative storage from the Cordova framework, the Secure Storage (see 5.4.1). This storage is designed for saving sensitive data. Although this plug-in works on device as well as on browser, the browser platform is actually not supported. Hence, the values stored are only stored in the local storage on the browser platform, making the secure storage without additions no good choice either. Thus, it is essential to encrypt data stored to both local storage and secure storage as a result of both storages being insecure on the browser platform. For this to work, an encryption service was needed. For this purpose, the library CryptoJS (5.4.4) was used which provides the ability to encrypt and decrypt data with AES.

The encryption offered by the library is accessed as shown in Figure 5.11.

Figure 5.11: Example for encryption and decryption with CryptoJS

```
import CryptoJS from 'crypto-js';

export class CryptProvider {

  constructor(private storage: SecureStorage) {}

  public static encrypt(data, key) {
    return CryptoJS.AES.encrypt(JSON.stringify(data),
      key).toString();
  }

  public static decrypt(data, key) {
    let bytes = CryptoJS.AES.decrypt(data, key);
    return JSON.parse(bytes.toString(CryptoJS.enc.Utf8));
  }
}
```

The secure storage with additional encryption can be accessed as shown in the code example in Figure 5.12.

Figure 5.12: Accessing the Local Storage with encryption

```
export class StorageProvider {

  constructor(private storage: SecureStorage) {}

  public storeValue(key:string, value:string) {
    this.storage.create('storage').then(
      (secureStorage: SecureStorageObject) => {
      let cipheredValue = CryptProvider.encrypt(value);
      secureStorage.set(key, cipheredValue);
    });
  }

  async getValue(key: string) {
    let data: string;
    await this.storage.create('storage').then(
      async (secureStorage: SecureStorageObject) => {
      let cipheredData = await secureStorage.get(key);
      await data = CryptProvider.decrypt(cipheredData);
    });
    return data;
  }
}
```

Considering that the secure storage is used asynchronous, the application has to wait until data is retrieved before returning. This is achieved by marking the ***getValue*** method and the created storage as async and then using the await command.

The biggest disadvantage coming with the use of secure storage and local storage is the limited amount of possibilities offered by key-value storages. But taking the limited possibilities offered for the browser module into account, this kind of storage seemed like the most useful tool.

### 5.3.5 Synchronising Local Data With Server

As already explained in 3.1.6, it is necessary to store most of the user's data inside the application for offline usability. Since this opposes the data to possible data inconsistencies, it is necessary to synchronize the user's data between the application and the back-end.

At the current state, crucial actions are disabled in offline mode for this purpose. These actions being finishing lessons, changing personal information including password and registration. After a successful login, the application will synchronize all user data including interventions, lessons, question elements and answer-sheets for each lesson of the user. Additionally, the application will synchronize a specific resource if the user tries to access it.

To prevent the application from loading all data again if it is unchanged on the back-end, the usage of conditional requests in form of HTTP e-tags was introduced. For further information, see the RFC documentation provided in [24]. These requests allow the application to check if data was changed instead of directly requesting the resource. The back-end saves an e-tag for each of its resources which is changed each time the resource itself is changed. After a successful get-request by the client to a resource, the back-end will add the resource's e-tag to the response. The client can now save it himself inside the application and add the e-tag and a HTTP if-modified header to the next get-request for this resource. This will result in the back-end comparing his own saved e-tag and the one he received from the client. If they match, the back-end will only respond with an HTTP 304 not modified response instead of sending the resource. The client will now know that nothing has changed and his locally saved data is up-to-date without the need to transmit the whole resource. If data was changed, the back-end will simply respond with a 200 HTTP Response and transmit the resource.

### 5.3.6 Auto-Login After Offline Login

Given that the user should be able to use the application offline, a local account system was needed as described in 3.1.6. For this to work, the user must have the ability to log into his account while no internet connection is active. Taking into account that the application will always try to communicate with the back-end if possible and that the user will not have a valid token due to the offline login, the application will try to send requests with no token provided if the internet connection is retrieved after the offline login.

This will result in all requests being answered with a 403 HTTP-Error-Response since authorization is needed and no token is present and would lead to a state of incapacity to do anything for the user. To encounter this problem, the application saves the users password encrypted inside the secure storage until the first request with internet connection is handled. For this purpose, Angular's HTTP interceptor is used again. The interceptor will detect when no token is present and internet connection is active. If this is the case and a password was stored by the offline login, the application will queue the request, decrypt the password and send a login request to the server with this data. As soon as the password is read from the secure storage, it is immediately deleted.

This request will retrieve a valid token from the back-end and will use this as authorization for future requests.

### 5.3.7 Injections

For the purpose of higher modularity and performance, Angular and thus Ionic uses a coding pattern called Dependency Injection (DI). These dependencies are external functionality a class needs to provide its own functionality. Dependencies in Angular are usually injected when the class which uses them is instantiated. For further explanations, see the documentation provided in [2].

Although this pattern has its advantages, there is one big disadvantage that has to be considered. By using DIs, the developer has to be cautious of not generating circular dependencies, meaning two classes cannot depend on each other.

As a result to this property, the developer has to build each different component, page or provider (see 4.3) carefully to not create a circular dependency. This is no problematic issue but has to be taken in account while constructing an application with Angular and thus with Ionic.

### 5.3.8 Ionic Local Notifications

To set weekly reminders for each unfinished lesson of a user, Ionic's Local Notification plug-in was used. This enables the creation of timed notifications while the app is either opened or closed. For this purpose, the plug-in uses a defined Provider (4.3.3) which creates and shows these notifications at a given time.

The challenge while working with this plug-in was the correct definition of the notifications, which should be created. The plug-in offers a great amount of possible options, but some of them did not seem to work as intended. This might be a result of the plug-in being defined for Cordova and not specifically for Ionic. Ionic wraps this plug-in for its own usage. Hence, the given documentation did not fully cover the usage of the plug-in. Some documented functionalities did simply not work as indented. This led to the need for a trial and error process to get the plug-in to create the notifications as intended.

### 5.3.9 Offline Availability Of Media Components Of Lessons

Local saving is mainly solved by using JSON and the secure storage for this application. Although this does work well for all user data including personal information, interventions, lesson etc, this is no solution for media elements, as this type of data cannot be parsed into a string without further addition. Furthermore, the media elements have to be saved on the local device as files to be able to display them on the GUI, which is achieved by binding these files to HTML tags.

For the media components to be available in offline mode, the application needs to be able to download a media file from a given URL provided by the lesson, save it on the device as file and then bind this file to the corresponding HTML tag. The first challenge faced was to download the file accessed through the URL.

Angular's HttpClient used for HTTP requests to the back-end offers a possibility to retrieve data from a given URL as BLOBs. BLOB is the abbreviation for Binary Large Object and embodies a binary data object, such as a picture or an audio file [23]. After retrieving the BLOB through the HTTP request, it needs to be converted to a file and then saved on the device. Ionic provides this feature through Cordova's "File" plug-in which offers the ability to handle the creation, deletion and usage of locally saved files and directories. This plug-in allows Ionic to create a corresponding file for the retrieved BLOB. After saving the file, the File plug-in can then be used to create an URL from the saved file which is only valid inside the application. This URL can then be used to bind the file to the corresponding HTML tag by settings its resource to the URL.

## 5.4 Plug-ins

This section will define additional plug-ins that were used for the application. These provide already implemented functionality to extend the framework.

### 5.4.1 Secure Storage

This plug-in provides the same functionality as Ionic's local storage but uses a secure storage. However, this only applies to the Android and iOS, thus making it necessary to encrypt all data to ensure data security on the browser platform, making the usage of this plug-in more an addition to mobile data security rather than solving the actual issue. The plug-in can be found on GitHub [8].

### 5.4.2 Spinner Dialoge

To provide additional data integrity and usability, the Spinner Dialogue plug-in can be used. This is a very simple plug-in for mobile devices making it possible to show a loading screen. As a result of the amount of data needed to be exchanged between application and back-end being relatively high, the synchronisation of the mentioned data will take some time. During this time it is essential to inform the user about the synchronisation.

However, this plug-in is not supported on the browser platform. To provide the same functionality for the browser, a variation of the Spinner Dialogue was implemented. This variation runs on all platforms but is visually not as appealing as the Spinner Dialogue. Hence, the Spinner Dialogue was used for mobile devices and the self-created variation for the browser module. This plug-in was published through GitHub [10].

### 5.4.3 NGX-Translate

To meet the requirement of multilingualism as defined in 3.1.4, the ngx-translate plug-in [12] was used. This allows the translation of defined keywords into language dependent values, thus providing the ability to create a multilingual application.

For this plug-in to run, a language JSON file is needed to be written for each language to be covered. All needed translations are identified by a key. For each key there is a value for each language. To demonstrate and clear-up the usage of the plug-in, the following will provide a small example.

Presuming that the application will cover English and German and the word "implementation" will be used, it is necessary to create two language files, called "en.json" and "de.json", one for each covered language. For additional languages, additional files have to be created.

Before being able to translate the word "implementation", a key is needed which will identify which word to translate. For the given word the key "IMPLEMENTATION" will be used. For the translation to work, the defined keyword and the corresponding translation for the language have to be added to the language files. The entry for the english language file is defined as shown in Figure 5.13,

Figure 5.13: English language file

```
"IMPLEMENTATION" : "implementation",
```

while the entry for the german language file is defined analogue (Figure 5.14).

Figure 5.14: German language file

```
"IMPLEMENTATION" : "Implementierung",
```

For the different translations to show, the application can now use the keyword and the translation command provided by the plug-in. If the user chooses English as language, the translation module's language setting will be set to "en", leading to the application reading from the "en.json" file. It is crucial that the module's language setting matches the corresponding file name.

By using the module's translate command with a given key word, the translation service will now read the corresponding value for the key from the set language file and will retrieve the found translation value. In this case, the service will retrieve "implementation" if the language is set to "en" and "Implementierung" if set to "de". Any missing key-value pair tried to be translated will result in the key to be displayed.

### 5.4.4 BcryptJS And CryptoJS

For the purpose of encryption and data security, two Java Script libraries were used. BcryptJS [9] is able to generate hashes for given parameters and supports salting of these hashes. Hashes in cryptography are encryptions which can not be decrypted again. This implies that it is not possible to get the plaintext password out of the hashed password-value. This makes it a very strong tool as encryption for user passwords. The only purpose of passwords is to authorize the user. With BcryptJS, it is not necessary to save the plaintext password of the user in the application.

Instead of decrypting the saved password and comparing it to the value that was typed in by the user, the entered password gets encrypted as well. BcryptJS can now check if both hashes have the same value without the need to decrypt the saved password. This provides a strong security for user passwords since no possibility is provided to read the plain text password of the user out of the application, even by knowing the encryption function.

In contrast to BcryptJS, CryptoJS [11] does not hash given values but encrypts them using AES, meaning that data encrypted with CryptoJS can be decrypted again. Both functions are provided by the CryptoJS plug-in. Given that most saved data has to be readable by the application, it does not make sense to use hash functions since this would imply that, once hashed, cannot be decrypted again. Thus, bcrypt is used for saving passwords and CryptoJS is used for saving data like interventions or lessons.

# 6 Requirements Comparison

Given the final outcome of the application, this chapter will reflect the requirements defined at the beginning 3 and compare them to the implementation.

## 6.1 Functional Requirements

1. **Cross-Platform: Browser, iOs, Android**
   To meet this requirement, the cross-platform framework Ionic was chosen as described in chapter 4.2. Ionic provides support for all three defined platforms, hence meeting this requirement, although the browser platform might have its limitations compared to iOS and Android. However, all requirements met for mobile devices were also met for the browser.
2. **Basic Account-System**
   Since basic account-systems usually do not differ greatly between different software systems, the implementation of this requirement was very straight-forward. However, the need for a local account system added more complexity to it. Nevertheless, the basic account system could be implemented as intended.
3. **Account Administration**
   Just as the basic account-system, this functionality is common in terms of software developing and could thus be implemented without further delay.

4. **Multilingualism**
   To provide full multilingualism, the application needed two different implementations for this topic. The first being the multilingualism of the application itself, including buttons, labels, headlines and texts. For this purpose, the plug-in ngx-translate was used as described in 5.4.3. The second implementation is related to the data loaded from the server. Content of interventions and lessons cannot be translated with ngx-translate but have to be loaded accordingly from the back-end. This already provides the ability to load the corresponding data in the desired language.

   Multilingualism application wide was achieved by providing the ability to set the desired language for both the application as well as for the data loaded from the server.

5. **Authorization And Authentication**
   The process on how authentication and authorization works in this software system and how it was finally handled was described precisely in Chapter 5.3.3, hence there is no need to cover this topic again. However, the requirement could be met as intended.

6. **Offline-Usability**
   Taking into account that data is stored locally and thus the system can be used while no internet connection is active, this requirement is met.

7. **Displaying Interventions, Lessons And Answer-Sheets**
   The application provides a view for all of the listed aspects. Interventions with their corresponding lessons can be viewed within the Home- and Intervention Overview Page, Lessons can be viewed more detailed on the Lesson Page and answer-sheets have their own listing for each lesson and a possibility to be displayed completely within the Lesson Page. Thus meeting the requirement as defined.

8. **Filling in Lessons**
   Since this requirement marks the core of the application, this was the application part the most effort was used on. By trying to provide an appealing and easy understandable view of all lesson elements for the user, this requirement was met. However, each user might have a different taste concerning the presentation of the lesson elements and might or might not find the view appealing. In addition to that, the view has not been tested on different users yet but only by the developer. Future developers might want to change the view if users appear to dislike it.

9. **Quick-Saving Answers**
   Quick-saving answers for all lessons was implemented successfully, however, for data integrity reasons quick-saved answers are deleted if the user changes the language of the interventions and lessons loaded from server. Future developers might want to extend this feature.

10. **Reminder**
    After having used enough time to figure out how the Local Notification plug-in (5.3.8) works as intended for ionic, it was possible to meet the requirement of weekly reminders for each unfinished lesson. This includes notifications while the app is opened on the device and while it is closed. However, the browser platform does not support the usage of notifications, resulting in the requirement only being met on Android and iOS.

## 6.2 Non-Functional Requirements

As non-functional requirements do not describe features and functionalities of the application, no concrete statements can be made if the application meets these requirements or not. Given that no testing with real users could have been done yet, the only statement that can be made is a personal assessment concerning these requirements.

1. Security
   By using BcryptJS, CryptoJS and the secure storage the intention was to build an application as secure as possible. However, the need to save a encrypted but not hashed user password after a successful offline login might mark an improvable part in terms of confidentiality. Additionally, the missing ability of the browser to use secure storage and only use the local storage is not desirable.
2. Usability
   The application was implemented with the goal of making the application as self sufficient as possible. This was tried to be achieved by building an easy structure of the app, using Ionic's self-defined html elements as introduced in 4.2 to provide visually appealing page elements and using additional CSS to support the user's understanding.
3. Maintainability
   The modularity given through Ionic's basic application structure enhances the maintainability of this application. Additionally, it eases the addition of new features by simply adding more modules or changing existing ones.
4. Modularity
   As a result of Ionic's basic application structure, modularity is already at a high level. This structure is described more precisely in 4.3. Each page, provider and component used is a stand-alone module that can the exchanged, altered or removed.

# 7 Conclusion

This chapter concludes the work by providing a small summary, a reflection of made decisions and a short outlook on how the system might be used and expanded in the future.

## 7.1 Summary

As no fitting software system is provided due to missing flexibility and missing functionality for the usage of KLIPS, a software system consisting of four different parts was planned. The goal for this work was the implementation of the Patient-Module, being one of the four software parts. To ensure that the planned application for the Patient-Module could provide all features needed, a list of requirements was formed. However, the application as planned by KLIPS is very future orientated in terms of functionality and used technology. As the implementation of the desired functionalities is not realistic for the amount of time given to implement, the requirements were shortened to provide a first version covering the most important and basic features which can be expanded in the future.

The application was then defined to be a cross-platform application to meet the desired concept of IBIs. For this purpose, the cross-platform framework Ionic was chosen to realize the application on mobile devices as well as on browsers. Before implementing the needed features with the given framework, it was essential to provide a basic design for the application, given mainly through the software architecture as provided by Ionic to ensure the outcome of the application to meet the desired requirements. Additionally, the process of the implementation was planned before and used existing approaches known from software engineering. These including issue tracking and the agile development method SCRUM.

After having implemented the application itself, the final outcome was compared to the initially defined requirements. Fortunately, these could be met accordingly. Since the desired software system as requested from KLIPS features more functionality than this first version of the application provides, it is much likely that it will be expanded in the future.

## 7.2 Reflection Of Decisions And Technology

Many decisions made while designing the application were based on given information gained through documentations and the experience provided by other developers, but not on the own experience. Hence, after having implemented the application with the chosen technologies, it might be interesting to reflect the decisions made beforehand. This Chapter reflects the design choices made before the implementation.

### 7.2.1 Ionic - Angular

After using Ionic as framework, Ionic feels like a strong tool for cross-platform development. This is mainly achieved through the usage of Angular as well as a easy understandable basic project structure. By providing already defined HTML elements with different styling for each platform, Ionic offers a helpful tool for easy and fast creation of views. This relieves the developer and takes off the need to define view elements for the different platforms which is usually a time-intensive task. Furthermore, the usage of Angular and its modules such as HttpClient and HttpInterceptor offer great possibilities especially in terms of client-back-end communication with the HTTP protocol. In case of this application, these modules take care of data exchange between the application and the back-end. Additionally, Ionic provides a modular application structure for higher modularity and maintainability without restricting the developer in his possibilities. This structure is given from the start of a project, hence maintaining modularity during the implementation process is quite simple. Finally, Ionic provides a great amount of possible plug-ins that can be used, not only from Ionic itself but also by wrapping Cordova. This allows the usage of Cordova's as well as Ionic's own plug-ins.

Although offering many tools and advantages, Ionic also has its downsides. As described in 5.3.7, the usage of DIs comes with the advantage of high modularity and performance, but requires a careful construction of the different modules to prevent circular dependencies. Also, some plug-ins from both Ionic and Cordova do not work on the browser platform. This might lead to the need to implement already existing functionality again to use them on the browser platform. Having not worked with either Ionic or Angular before, the amount of initial effort that has to be made was relatively high. Like DIs, this is not an actual problem but has to be taken in account when working with Ionic and Angular the first time.

Taking the stated properties into consideration, I would recommend the usage of Ionic and would use it again. Although having trouble at the start of new functionality due to the small amount of knowledge temporarily, Ionic is a strong tool for cross-platform development at the end.

### 7.2.2 Ionic's Local Storage And Secure Storage

Ionic offers powerful possibilities for storage purposes. However, the browser platform does not support most of them. This led to the usage of the secure storage and the local storage instead of a database. Hence, the data structure for saved data is JSON in key-value storages. This scales bad for large data volume. At the current state, this is no big problem but might get confusing with increasing amount of data.

Since the requirements define offline usage for browser platform too, the only other option is a database mock for the browser platform to be able to use real databases on the device. With the knowledge of the application being most likely extended in the future, this might have been a better solution than the usage of key-value storages.

### 7.2.3 Issue Tracking

As described in 5.1.2, the observable, formal issue tracking was switched out for better team management. Although this worked out really well and saved time for myself, the possibility of tracking these issues for other parties involved in the software system such as other developers or the supervisor got lost. Since meetings with all other team-members and the supervisor were held weekly, this was not needed compulsorily but would have increased the overall observability.

Considering these factors, it would have made sense to use any kind of formal and observable issue tracker for this application.

## 7.3 Outlook

This section addresses the future work that might or might not be implemented after the work for this project is done. This will most likely be addressed by other developers. The following topics already cover future requirements collected by the institute meeting (see 3) partially.

### 7.3.1 Local Storage

With expanding amount of data to be stored on the client side, it should be considered to remove the ability for offline-usage for the browser platform since this platform usually has connection to the internet. This would offer the possibility to use stronger and better scaling storages than key-value storages. Also, the security for the stored data can be secured more easily.

If this is no possible option, the usage of a database mock for the browser platform might help the overall observability and understanding of the structure the storage uses.

### 7.3.2 Expanding Reminders

Reminders currently are fixed notifications for mobile devices which will appear each week for each unfinished lesson. The possibilities for this feature can cover much more use cases and might be expanded in the future. Users might want to personalize their reminders in terms of how often and when they appear.

Since the notifications used for these reminders only work on devices and not on the browser platform, the provision of the possibility to set not only device notifications but reminders in form of emails or any other not platform dependent reminder should be considered.

### 7.3.3 Communication Of E-Coach And User

Considering that the software system will most likely want to support guided IMIs, a possibility for communication between e-coach and patient might be needed. For this purpose, some kind of textual chat system will most likely be wanted. Even for non-guided IMIs is might be interesting for the user to receive feedback from the corresponding e-coach. Since this will run on mobile devices, access and usability of the chat system should be easy and self-sufficient.

### 7.3.4 Conditional Components

This feature is related to the process of filling in lessons. For usability and a higher possibility provided by lessons it is desired that elements can depend on each other. This should be implemented through conditional contents. E-coaches might want to only show elements of a lesson if specific conditions are met, for example if a specified question element was answered with a specific answer. This feature would provide more possibilities for extended lessons and interventions.

### 7.3.5 Offline Mode

For offline usability, the application checks if it is connected to a network. If not, it will go into offline mode, meaning it will prevent the sending of all requests and will work with the data saved locally until an active connection is retrieved.

The challenge is to actually detect missing internet connection instead of missing network detection. The application will try to send requests if it is connected to a network but not to the internet due to missing internet connection of the network. A reasonable detection for missing internet connection is needed for the future.

### 7.3.6 Personalization

For the purpose of user comfort while using the application, users might want to personalize the application to their personal preference. This might include different backgrounds, profile pictures or any other possible personalization option.

# Bibliography

[1] Gerhard Andersson and Nikolai Titov. “Advantages and limitations of Internet-based interventions for common mental disorders”. In: *World Psychiatry, Official Journal of the World Psychiatric Association (WPA)* (Feb. 2014). URL: `https://www.onlinelibrary.wiley.com/doi/full/10.1002/wps.20083`.

[2] Angular Documentation. *Dependency Injection in Angular*. [accessed 25-March-2019]. 2019. URL: `https://www.angular.io/guide/dependency-injection`.

[3] Apache. *Apache Cordova*. [accessed 28-March-2019]. 2019. URL: `https://www.cordova.apache.org/`.

[4] Drifty Co. *Ionic Framwork*. [accessed 16-March-2019]. 2019. URL: `https://www.ionicframework.com/`.

[5] Drifty Co. *What is Ionic Framework?* [accessed 5-April-2019]. 2019. URL: `https://www.ionicframework.com/docs/intro`.

[6] Facebook. *React-Native*. [accessed 16-March-2019]. 2019. URL: `http://www.reactnative.com/`.

[7] Frank Kargl and Elmar Schoch. “Kapitel 1: Grundlagen - Security in IT-Systems”. Lecture at Ulm University, Wintersemester 2017/2018. 2017.

[8] GitHub repository contributors. *SecureStorage plugin for Apache Cordova*. [accessed 5-April-2019]. 2019. URL: `https://www.github.com/Crypho/cordova-plugin-secure-storage`.

[9] GitHub repository contributors. *bcrypt.js*. [accessed 5-April-2018]. 2018. URL: `https://www.github.com/dcodeIO/bcrypt.js`.

[10] GitHub repository contributors. *cordova-plugin-native-spinner*. [accessed 5-April-2018]. 2019. URL: `https://www.github.com/greybax/cordova-plugin-native-spinner`.

[11] GitHub repository contributors. *crypto-js*. [accessed 5-April-2018]. 2019. URL: https://www.github.com/brix/crypto-js.

[12] GitHub repository contributors. *@ngx-translate/core*. [accessed 5-April-2018]. 2019. URL: https://www.github.com/ngx-translate/core.

[13] Google. *Android*. [accessed 14-March-2019]. 2019. URL: https://www.android.com/.

[14] Google. *Angular Framework*. [accessed 16-March-2019]. 2019. URL: http://www.angular.io.

[15] Google. *TypeScript*. [accessed 16-March-2019]. 2019. URL: http://www.typescriptlang.org.

[16] Jibitesh Mishra and Ahsok Mohanty. *Software Engineering*. 1st ed. Pearson India, 2011. Chap. 9.6. ISBN: web-ISBN-13: 978-81-317-5869-4.

[17] Josephine Königbauer et al. "Internet- and mobile-based depression interventions for people with diagnosed depression: A systematic review and meta-analysis". In: *Journal of Affective Disorders* 2.1 (Aug. 2017), pp. 841–881.

[18] Margaret Rouse. *cross-platform mobile development*. [accessed 13-March-2019]. 2019. URL: https://www.searchmobilecomputing.techtarget.com/definition/cross-platform-mobile-development.

[19] Mark Edward Soper. "CompTIA Fundamentals+ FCO-U61 Cert Guice, First Edition". In: 1st ed. Pearson IT Certification, 2018. Chap. 21. ISBN: ISBN-13: 978-0-13-525889-7.

[20] *Mobile Usability*. 1st ed. Publisher website: https://www.mitp.de. Augustinusstr. 9a, 50226 Frechen: mitp, 2013. ISBN: 978-8266-9503-2.

[21] Mohd Shahdi Ahmad and Nur Emyra Musa and Rathidevi Nadarajah and Rosilah Hassan and Nor Effendy Othman. "Comparison between android and iOS Operating System in terms of security". In: *2013 8th International Conference on Information Technology in Asia (CITA)* (2013).

[22] OAtuh. *JWT*. [accessed 14-March-2019]. 2019. URL: https://www.jwt.io/.

[23] Oracle Corporation. *11.4.3 The BLOB and TEXT Types*. extract of MySQL 5.7 Reference Manual, accessed 28-March-2019. 2019. URL: https://dev.mysql.com/doc/refman/5.7/en/blob.html.

[24] R.Fielding and J.Reschke. *Hypertext Tranfer Protocol (HTTP/1.1): Conditional Requests*. RFC 7232. RFC Editor, 2014, p. 1. URL: `https://www.rfc-editor.org/rfc/rfc7232.txt`.

[25] Roman Pichler. *Scrum: Agiles Projektmanagement erfolgreich einsetzen*. 1st ed. dpunkt.verlag, 2007. ISBN: ISBN-13: 978-3898644785.

[26] Sahoo Saddicha et al. "Online interventions for depression and anxiety - a systemativ review". In: *Health Psychology and Behavioral Medicine* 223 (Dec. 2017), pp. 28–40.

[27] Vadzim Belski. *Ionic vs Pure Cordova: Three Reasons Ionic Wins*. [accessed 15-March-2019]. 2019. URL: `https://www.jotform.com/blog/ionic-vs-pure-cordova-97503/`.

Name: Florian Matthias Haschka Matrikelnummer: 870351

**Erklärung**

Ich erkläre, dass ich die Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Ulm, den ..........................................................................

Florian Matthias Haschka