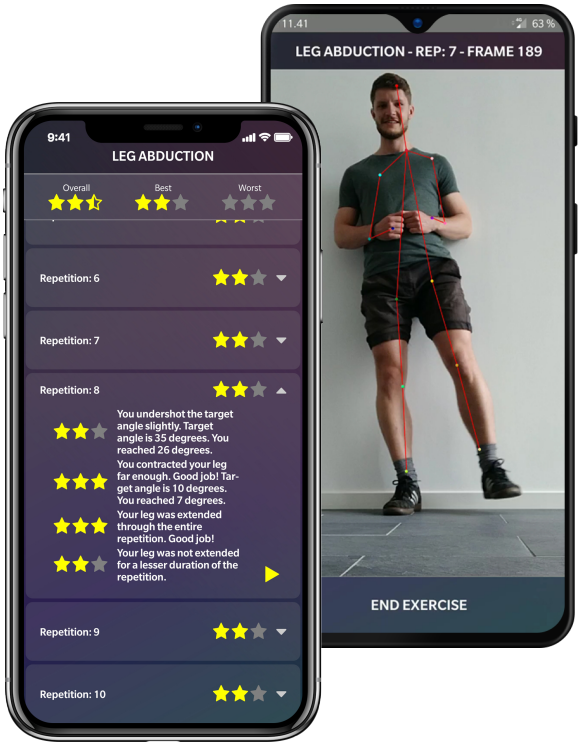
# FormAssist

**Proper form, steady recovery!**





**Group SW810F19**

Kasper Dissing Bargsteen

Jonatan Groth Frausing

Kristoffer Magill Nash

Niclas Jon Sommer

Peter Vergerakis

Joachim Valdemar Yde



March 8, 2021

Aalborg University

Software, 8. semester

**Computer Science**
Aalborg University
http://www.aau.dk

**AALBORG UNIVERSITY**
STUDENT REPORT

**Title:**
FormAssist - Proper form, steady recovery!

**Theme:**
P8 Project

**Project Period:**
SpringSemester 2019

**Project Group:**
SW810F19

**Participant(s):**
Kasper Dissing Bargsteen
Jonatan Groth Frausing
Kristoffer Magill Nash
Niclas Jon Sommer
Peter Vergerakis
Joachim Valdemar Yde

**Supervisor(s):**
Eike Schneiders

**Copies:** Online on Digital Exam

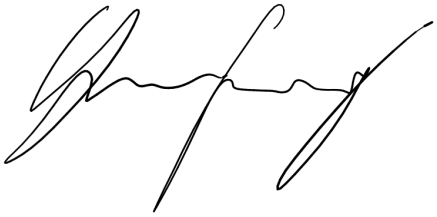
**Page Numbers:** 107

**Date of Completion:** March 8, 2021

**Abstract:**

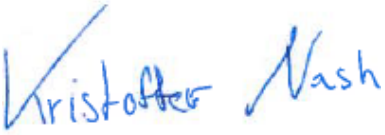
We set out to create a smartphone application which can aid patients of physiotherapists in completing their prescribed exercises. To this end, we built a cross-platform application for Android and iOS using React Native. *FormAssist* employs a pre-trained pose estimation model to estimate the pose of a patient and, subsequently, detects the repetitions being performed and evaluates them. Real-time and post-performance feedback is provided to the patient in order to correct possible mistakes. *FormAssist* has been built with a strong emphasis on extensibility in order to extend its current repertoire of exercises in the future. Most features of the MVP have been implemented accordingly, though we have identified a number of possible avenues for future development, one of which being the notion of motivation through gamification elements.

# Preface



Kasper D. Bargsteen
kbargs15@student.aau.dk

Jonatan G. Frausing
jfraus14@student.aau.dk



Kristoffer Magill Nash

Niclas Jon Sommer

Peter Vergerakis

Joachim Valdemar Yde

# List of Figures

# Contents

# Chapter 1

# Introduction

In Denmark, as of 2016, 61% of the population aged 16 and above regularly exercise [1]. Of those who exercise, 21.5% experience an injury within a year—the vast majority of which from running, football, and weightlifting [2]. As evident by these statistics, Danes are actively and regularly exercising by themselves or with friends and family in local sports clubs as well as fitness centres. Globally, the total number of members in health and fitness centres has increased from about 120 million in 2009 to roughly 174 million in 2017 [3]. It is, therefore, safe to say that exercising is becoming a regular activity for an increasing amount of people.

As the number of people regularly exercising increases, as does, inherently, the total number of sustained injuries. As an example, in weightlifting, which is one of the main sources of injuries for Danes [1], many newcomers are at an increased risk of overestimating themselves and their abilities, or, equally troublesome, underestimating the difficulty of a given exercise. Incorrectly performing exercises, as well as accidents pertinent to the performance of said exercises, can, depending on the specific circumstances, induce mild to severe injuries. Sustaining any injury from physical activity can negatively impact not only the enjoyment of participation—and thereby reduce the long-term health benefits associated with physical activity which in turn provokes a whole host of other issues—it can also compromise function later in life [4]. In general, sports injuries are most commonly caused by overuse, or through the application of force that is greater than the body part can structurally withstand [5]. Injuries are broadly categorised as being either acute or chronic. An injury which occurs suddenly—e.g. an ankle being sprained following an awkward landing when running—is considered an acute injury, whereas injuries caused by repeated overuse of specific muscle groups or joints are considered chronic. A poor technique or structural abnormalities may, furthermore, contribute to the development of chronic injuries [6].

The risk of sustaining an injury is not confined to those engaging in physical activity, however. A majority of the workforce in Denmark is employed in sedentary work [7, 8]. This growing percentage of the population is at an increased risk of sustaining an injury, often times as strain, as a direct consequence of their hours at work [9, 10, 11]. Whilst the specific injuries ordinarily associated with sedentary work might not overlap those commonly sustained during physical activity, the impact they

have on the lives of those who suffer can be just as serious. In 2000, an American study revealed that musculoskeletal disorders—such as repetitive stress injury, carpal tunnel syndrome, and tendinitis [12]—accounted for more than $15 billion every year in worker's compensation costs and approximately 34% of all lost-workday injuries and illnesses [13]. Similar high societal costs in Denmark were documented in a 2013 study which revealed how the same group of injuries carry an estimated annual cost of 20 billion DKK [14].

Regardless of how an injury is sustained—be it through physical activity, sedentary work, or something else entirely—many seek help to alleviate, or heal, the given injury. Where those who sustain acute injuries during physical activity are likely to seek immediate medical attention—the above-mentioned Danish study notes how approximately 100,000 annual emergency room visits are due to injuries sustained during physical activity, for instance—followed by rehabilitation, the course of action differs for those suffering from chronic injuries who are more likely to go through non-emergency channels. In either case, a physiotherapist can provide professional insight into what might have caused the injury, as well as how to successfully recuperate. A treatment usually involves meticulous work on and around the affected area(s) seeking to isolate the issue(s). Once identified, appropriate exercises for the patient to perform at home are prescribed. The performance of unsupervised exercises as instructed by the physiotherapist is, consequently, an important aspect of properly recuperating the sustained injury. However, a study has shown that upwards of 54% of clients perform the prescribed exercises incorrectly or not at all [15]. This can potentially result in a worsening of the injury.

In the context of bettering an injury, be it acute or chronic, we investigate the type of injuries that physiotherapists typically deal with and how such injuries are treated. We focus on this aspect for a couple of reasons. The first reason being that both those who require assistance beyond what is offered by an emergency room following an acute injury, as well as those suffering from chronic injuries, are likely to seek the assistance of physiotherapists in some capacity. The second reason being that whilst somehow preventing injuries from occurring in the first place would be ideal, attempting to do so unarguably constitutes an insurmountable task. It, consequently, makes sense to focus on what is ultimately a confined procedure and investigate which, if any, complications may arise during the treatment of a typical injury, as well as whether treatments are currently aided by any tools or techniques relying on technology.

To this end, we pose the following initiating problem for further analysis:

*How can we gain a better understanding of the course of a treatment following an injury?*

# Chapter 2

# Problem Analysis

In the previous chapter, we noted how both the amount of people subject to sedentary work, as well as the amount of people regularly engaged in physical activities, is significant and increasing. As a consequence, both groups face a greater risk of injury—something which, as also established in the introduction, can lead to a number of serious implications carrying both high individual and societal costs. We, moreover, noted how we wish to advance our understanding of the various, most common injuries, as well as their associated treatments as recommended by physiotherapists.

As this is a new problem domain for all members of the group, we conduct two semi-structured interviews with professional physiotherapists who agreed to speak with us. These interviews will, together with relevant papers, lay the foundation for our understanding of the problem domain. It should be noted that we opt for this specific interview technique because we are primarily interested in attaining a broad understanding in an explorative manner. We, therefore, pose the same eight, open-ended questions to each physiotherapist. These questions are intended to guide the conversation and encourage an honest discussion—the reader may find the questions attached in appendix A. We, moreover, wish to avoid making any assumptions as well as avoid deliberately steering the conversation in any specific direction. Whilst we rely on the abovementioned, open-ended questions, we do allow ourselves to pose ad-hoc questions based on the answers provided by the interviewee. We could have opted for not bringing any questions at all in favour of an unstructured interview, but doing so would come with the inherent risk of not attaining an adequate understanding of the domain, and unarguably relies more heavily—if not exclusively—on the interviewer being good at producing high quality questions extemporaneously. A fully-structured interview was, similarly, not a good option given our lack of understanding of the problem domain.

We discovered through our interviews with physiotherapists Mads Bomholt Jensen and Lars Henrik Dahlberg that people most commonly visit physiotherapists for one of two reasons, namely (1) rehabilitation following surgery, and (2) following injury, or injuries, sustained from having excessively performed one or multiple physical actions during either physical exercise or sedentary work. This is very much in line with what the statistics introduced in the previous chapter suggested. It should be accentuated that whilst most of Mr. Jensen's patients visit him for rehabilitation purposes following

surgery, and Mr. Dahlberg mainly focuses on work-related shoulder and neck problems, they both identified the above differentiation independently.

In either case, the course of a treatment is often a combination of (a) in-person treatments in the form of massages and / or supervised training sessions with the physiotherapist, and (b) exercises designed to be performed unsupervised at home, or at the gym, over a period of time. Whilst a complete treatment is often twofold in that some parts are done in-person together with the physiotherapist, and other parts are subsequently done at home, it is usually only one of the two which pose a challenge. That is, there are usually no problems during the supervised sessions in which the physiotherapist can adjust and give advice in real-time as exercises are being performed. There are, however, some issues associated with the exercises which the patient has to perform unsupervised as also indicated by the study referenced in the previous chapter [15]. In general, there are two leading challenges associated with this for a subset of patients, namely (1) patients incorrectly perform their exercises, and (2) patients either do not perform their exercises at all, or perform them too infrequently.

Using the answers obtained during our interviews in conjunction with existing research [15], we devise the illustration shown in figure 2.1. This illustration encapsulates the most commonly identified causes for (1) exercises being performed incorrectly, and (2) exercises not being performed at all. Note how each trace leading to either one of the two consequences, highlighted in blue, should be read inwards from the outermost node where each edge translates to an implication arrow. It should, additionally, be noted how whilst figure 2.1 is comprehensive and seeks to account for most reasons, it should not be considered definitive.

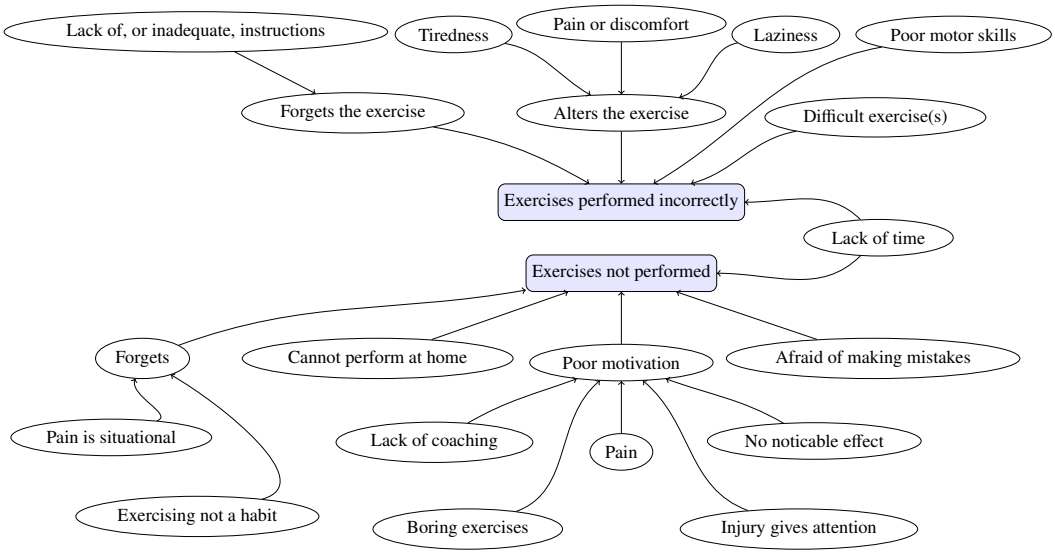



**Figure 2.1:** The leading causes of exercises not being performed, or exercises being performed incorrectly. The blue nodes represent the two consequences, whereas white nodes represent causes.

In an attempt to mitigate the issue of exercises being performed incorrectly, and to aid the supervised sessions, both physiotherapists employ a number of tools when designing treatments. One such tool are video cameras, which are used to record patients performing either a single repetition of an exercise or a sequence of repetitions. The physiotherapist can then subsequently replay the recording whilst describing and highlighting any potential issues to the patient. Mr. Jensen also noted how their clinic occasionally conducts experiments by supplying the patients with smartphone applications which include videos of the exercises they have to perform. Another tool employed by both physiotherapists is the utilisation of a large centralised database consisting of exercises specifically designed for physiotherapists. Examples of some of the exercises prescribed by both Mr. Jensen and Mr. Dahlberg can be seen in appendix B and C, respectively.

We, additionally, found that both physiotherapists provide thorough instructions on how to properly perform the various exercises—often in the form of videos or illustrations on paper—to aid the patients. However, as already established, and subsequently confirmed, by both physiotherapists, this is no guarantee that patients will necessarily perform their exercises correctly—a survey found that 19% of participants did not perform their assigned exercises at home because they had forgotten how to properly perform them [15]—nor is it any guarantee that patients will even perform their exercises in the first place. This introduces the risk of some patients performing exercises incorrectly for what might be several weeks before their next consultation—something which could possibly worsen their condition. Another leading reason for exercises not being performed is pure forgetfulness; the same survey concluded that 24% of participants did not perform their exercises because they forgot. However, the absolute majority of those who did not perform their exercises did not do so due to a lack of time. This sentiment also rings true in our interviews. An additional 25% of participants in the survey indicated "boring exercises" as the cause for their lack of adherence to the exercise plan provided by their physiotherapist.

Towards the end of the interview with Mr. Jensen, and during a closing conversation with Mr. Dahlberg, we discussed potential solutions to motivate patients to perform their exercises as well as how to increase the correctness of the performed exercises. Independently, they both suggested a solution based on a smartphone application as a suitable solution because it can bundle multiple types of feedback and reminders into a single system. It is important to realise that these two problems—patients forgetting to exercise, and patients performing exercises incorrectly—are of symptomatic character, and fixing them, therefore, requires an analysis of their causes—as per figure 2.1.

We could opt for focusing exclusively on helping patients with remembering when to perform their exercises by developing a smartphone application which pushes notifications at the appropriate intervals. We could also attempt to accommodate the portion of patients who indicated in [15] that they found their prescribed exercises boring. This could, for instance, be achieved by expanding an otherwise simple notification application with the incorporation of some form of positive feedback loop [16], or even various gamification elements[1] [17], to further motivate the patients. This, however, would not help the patients who actually perform their exercises, but do so incorrectly, nor would it help those who want to perform their exercises, but have forgotten how to perform them correctly. Both Mr. Dahlberg and Mr. Jensen provide their patients with materials to aid them with their ex-

[1] We will return to this in greater detail throughout section 3.1.1.

ercising at home. However, they both indicated that this is not always sufficient to prevent some patients from still inadvertently performing exercises incorrectly during unsupervised sessions. We, therefore, want to focus on both of these issues. In other words, we need to a develop an application which, in addition to providing timeful reminders, can aid the patient during their unsupervised sessions both from a motivational and correctness perspective. Correspondingly, we need to develop a system which has the functionality to accurately determine (a) whether a patient is performing an exercise, and (b) estimate whether the patient is performing the exercise correctly. To this end, we pose the following problem definition:

*How can a software solution assist patients of physiotherapists in performing their prescribed exercises correctly and at the assigned frequency?*

More specifically, our solution should consider how to address the following sub-problems:

- Motivate patients to perform their exercises.
- Detect whether the patients perform the prescribed exercises.
- Evaluate the correctness of the performed exercises.

# Chapter 3

# Solution Analysis

Having defined a problem statement, we now proceed to investigate related existing solutions by conducting a state-of-the-art analysis. We will also investigate the ecology—that is, the people, technologies, and products currently surrounding the patients—to understand who and what can be leveraged in terms of developing a sound solution. We will, conclusively, present a solution outline.

## 3.1 State-of-the-art

Our problem statement is twofold in that it addresses two independent concerns—that is, (1) how to motivate patients and help them remember to perform their prescribed exercises, and (2) how to assist patients in correctly performing their prescribed exercises. We refer to this as *motivation* and *proper form*, respectively. In either case, we will base our state-of-the-art analysis on the conducted interviews alongside relevant papers.

### 3.1.1 Motivation

As explained by Mr. Dahlberg, treatments are often heavily reliant on unsupervised exercises. It is, therefore, essential that the patient remains motivated to perform their exercises at home—indeed, a large portion of physiotherapists' time is spent together with patients concerning motivation in an coach-like setting (see appendix section C). The primary tool in the physiotherapist's toolbox towards ensuring exercises are performed is, therefore, conversing with and coaching the patient. It may seem strange that motivation is sometimes an issue given how patients either arrange an appointment with a physiotherapist themselves, or are recommended one by their doctor, and, therefore, should be motivated to undergo treatment. However, because a treatment's success is wholly dependent on the patient's engagement, it is, nonetheless, necessary. According to Mr. Dahlberg, maintaining patient motivation is a continuous process throughout the entire course of a treatment. He discusses subjects such as progress, pain, and any related challenges with his patients during every consultation. Depending on what is discussed during each consultation, the course of treatment can be adjusted accordingly. Mr. Dahlberg primarily uses statistics to emphasise the effect of different exercises

and to choose between said exercises when designing treatments. He does not, presently, utilise the statistics as a way of motivating his patients. It is worth noting how neither Mr. Jensen nor Mr. Dahlberg presently rely on technology in motivating their patients.

Another approach to motivation which has gained a lot of traction in both industry and research communities is the aforementioned *gamification* [17]. The essential idea of gamification is to make an arbitrary activity more interesting by introducing mechanics known from games—hence the name. Whilst gamification can be achieved through several intricate techniques, two general approaches emerge, namely (1) implementing a reward system on top of an existing activity without altering the activity itself—e.g. the online learning platform KhanAcademy [18], where you receive points for watching videos and completing quizzes—and (2) altering the activity itself in order to make it more enjoyable. The latter of the two is arguably more demanding as it requires the creators to determine which underlying elements make the activity interesting in itself and then emphasise those elements accordingly. A great example of this sort of gamification is the award-winning mathematics game series DragonBox [19], which focuses on discovery and experimentation as to introduce players to the basic processes of algebra without them realising that they are learning. Whether the first or second approach is employed, gamification has a measurable effect on how motivated people are to perform the given activity [17].

The concept of gamification is also applicable to sports. Indeed, the fitness community quickly recognised the power of gamification. Google and Apple with their Google Fit and Apple HealthKit products have both developed applications for mobile and smartwatches which track and reward the user for performing physical activities [20, 21]. These applications do not alter the underlying activity and, consequently, belong to the first category of gamification. Other examples of gamification being employed to encourage physical activity can be found in Nintendo's Wii Balance Board [22, 23] and Microsoft's Kinect Shape Up [24]. These both introduce the user to a virtual world in which the user has to, for instance, perform squats in order to move along a race track and outpace their competition. Instead of showing the virtual world on screen, it is also possible to further enhance the sense of immersion by employing virtual reality headsets such as the Oculus Rift [25, 26, 27]. All these are examples of products—both hardware and software—which employ gamification with the ultimate goal of motivating people to exercise by making physical activity fun.

### 3.1.2 Proper Form

There currently exists two leading types of solutions which help patients perform exercises using proper form, namely (1) descriptive tools which describe how to perform a given exercise, either via text, diagrams, video, or something else entirely, and (2) analytic tools which capture information about how a given exercise is performed—for instance by recording video, images, or motion data such as accelerometer output. Where the former is mainly employed prior to an exercise being performed—though instructions can certainly also be checked post-performance—the latter solution is entirely post-performance seeing how the recorded information is subsequently analysed either automatically by dedicated software, or manually by a specialist like a physiotherapist.

It is, moreover, not uncommon for physiotherapists to supply patients with a combination of all

the descriptive tools via a brochure—we have included a snippet of such a brochure in fig. 3.1—containing diagrams of the exercises alongside explanatory texts with references to videos of the specific exercises being performed correctly.


**Figure 3.1:** Part of an exercise brochure given by physiotherapists.

We, furthermore, researched technological solutions that aim to detect how well certain exercises are performed. One such example is the smartphone application *Rithmio Edge* which, in addition to the functionality featured in Google Fit and Apple HealthKit, provides detection of which specific weightlifting exercise is being performed coupled with functionality such as counting the number of repetitions [28]. This means that users need not specify every time a new exercise is being performed. It uses the accelerometer in smartwatches for detection, and provides visual feedback to the user on the screen. Rithmio Edge does not provide any analysis of the user's form. Kinect, on the other hand, which is also used in Shape Up, creates a 3D image in which each pixel, in addition to a colour, has a value describing the distance to the camera which can be used to provide estimation of where each body part is. This technique is called *body pose estimation* [29], and has successfully been employed to analyse how well exercises are performed [30, 31]. Indeed, in order to determine the correctness of a given exercise, any developed software solution must, inherently, be able to not only capture the exercise being performed, but also be able to distinguish between the body performing the exercise and everything else.

## 3.2 Solution Outline

As per our problem analysis, and subsequent solution analysis, we effectively face two independent issues which we want to solve. What both issues—motivation and proper form—share, however, is the fact that they both occur during unsupervised sessions. This means that the solution we develop should be accessible to as many patients as possible—something which inherently means that we can-

not rely on expensive or uncommon equipment. The prevalent state-of-the-art solutions introduced above are based on one, or a combination of, the following physical devices: Smartphones, smartwatches, Microsoft Kinect, virtual reality headsets, and computers with cameras. Of these devices, smartphones and smartwatches are not only commonly available ones, they are also most mobile in the sense that they do not depend on a specific setup—a Microsoft Kinect requires both a console or a computer and a stationary screen, for instance. Furthermore, among the solutions currently in use, body pose estimation, akin to what the Kinect achieves, seems like the best available, and most powerful, technique for evaluating proper form. That is, an accelerometer in a smartwatch is not necessarily able to determine whether an exercise has been performed correctly, but merely whether an exercise has been performed.

As mentioned, physiotherapists utilise video as part of explaining and instructing patients on how to perform exercises correctly. A solution which offers a similar type of assistance without the need for a physiotherapist to be present can potentially help patients maintain proper form during unsupervised sessions and, thereby, alleviate one of the primary issues. Of the available technologies, the only one which is reasonably mobile, and almost ubiquitously features a camera, is the smartphone. This is a must given that we need to be able to actually *see* what happens as an exercise is being performed. It is, additionally, a huge advantage that almost 80% of Danes already own a smartphone [32]. Of those who own a smartphone, approximately 53% own an iOS device, and about 46% own an Android device [33]. We will, consequently, base our solution on such devices. Our solution should allow the patient to record themselves performing an exercise, analyse the correctness of the performed exercises, and finally provide feedback to the patient.

In order to propose a solution which adequately addresses the highlighted issues throughout this section, we will develop a cross-platform smartphone application named *FormAssist*. We require a set of heuristics— akin to the videos physiotherapists refer some patients to—that can be employed to determine the correctness of a given exercise based on a recording of that exercise being performed such that appropriate feedback can be provided. We imagine that heuristics are mostly exercise dependent—that is, where it for some exercises may make sense to determine correctness based on distances between certain body parts, it may make more sense to compute angles between certain body parts for other types of exercises. Other possible heuristics determining correctness could be a patient's positioning throughout a given exercise, or perhaps the time certain movements take. Please note that we will discuss this in greater detail throughout our design and subsequent implementation section. Given that a patient presumably performs a number of repetitions of a specific exercise, we must be able to apply the heuristics on each individual repetition—something which by natural extension also requires *FormAssist* to be able to differentiate the start and the end of each repetition of the exercise in question. The core of *FormAssist* will, therefore, include functionality to:

- Generate push notifications when exercises should be performed.
- Detect and differentiate repetitions of an exercise.
- Provide feedback following the performance of a supported exercise.
- Provide some form of gratification when an exercise has been performed correctly.

Of the four points above, the first and fourth relate to motivation, whereas the second and third relate to proper form. We will now address each core functionality and explain in general terms how we imagine *FormAssist* will accommodate them.

### 3.2.1 Solution — Motivation

As identified in the previous section, forgetfulness and boring exercises are two common explanations for why patients do not perform their prescribed exercises. It should be noted that whilst a lack of time was identified as the single largest reason per [15], forgetfulness and boring exercises collectively account for a larger share. We believe push notifications will address forgetful patients by reminding them at appropriate intervals to perform their prescribed exercises. In order to also motivate the patients bored by exercises, it is relevant to consider how *FormAssist* may provide gratification when performing exercises. As mentioned in section 3.1.1, gamification of various tasks has proven to be an effective motivational tool. One way to employ gamification is to introduce the concept of achievements, rewards, or ratings for successfully performing some activity. We will, therefore, consider the possibility of having *FormAssist* feature rewards as an incentive for the patient to perform their prescribed exercises.

### 3.2.2 Solution — Proper Form

Since *FormAssist* ultimately has to accommodate patients who do not perform their exercises correctly, it must be able to provide feedback to the patient. That is, whilst recording oneself performing an exercise, and subsequently reviewing the video, is relatively accessible for most patients, determining what is correct and what is incorrect often requires domain-specific knowledge. This is why *FormAssist* must be able to provide constructive and accurate feedback based on predefined correctness heuristics. This can aid the patient in not merely identifying what went wrong, but also how to correctly perform the exercise in question. Providing feedback that is both helpful and intuitive is ultimately a task which requires consideration regarding *how* and *when* feedback should be provided. We consider the following two possibilities for the development of *FormAssist*:

- Provide feedback after an exercise has been performed.
- Provide feedback whilst an exercise is being performed.

Recording and subsequently evaluating the performance of all repetitions of a given exercise allows *FormAssist* to provide a patient with a comprehensive assessment of their overall performance. This can provide the patient with detailed information about their form. Indeed, textual feedback, or even video playback, would allow *FormAssist* to pinpoint and highlight exactly what went wrong as well as provide suggestions on how to correct the identified mistake(s). The patient can, moreover, in a post-performance setting take their time to properly process the feedback—something which means the amount and complexity can be high if necessary. Only supplying post-performance feedback has a number of shortcomings, however. It is easy to imagine a situation, for instance, in which a patient has performed $x$ repetitions of a given exercise, only to be told, after the fact, that every repetition was subpar—something which could be demotivating. Similarly, if a patient is correctly performing

an exercise, but do not perform enough repetitions—or perhaps too many repetitions—they would only be told afterwards.

Providing feedback whilst an exercise is being performed—that is, real-time feedback—addresses these shortcomings. Real-time feedback can aid a patient by allowing them to rely on *FormAssist* to assure them that certain conditions of an exercise are met. However, the amount and complexity of the feedback provided should be relatively low seeing how attempting to process a lot of detailed information whilst simultaneously performing an exercise would be cumbersome if not impossible. Thus, we must find an adequate balance in our design between real-time and post-performance feedback to best assist the patient. We must, additionally, determine how to best provide the feedback—be it in textual form, as visual or sound cues, or something else entirely.

### 3.2.3 Solution — Summary

In order to accommodate motivation, we must first and foremost, with the exception of push notifications, address the proper form aspect: In the scope of this project, it does not make sense to motivate patients to perform exercises if we cannot aid them in performing them correctly. As already established, exercises should be performed correctly to avoid, potentially, worsening the sustained injury. This means that we will start by focusing on developing the functionality allowing *FormAssist* to detect when a repetition has been performed, and then develop the functionality allowing *FormAssist* to provide relevant post-performance and real-time feedback.

Whilst we envision a release candidate of *FormAssist* to include both motivational factors, such as gamification elements and push-notifications, and proper form, we need to account for the time restrictions inherent to a semester project. Additionally, gamification is in and of itself an advanced topic in which we have no prior experience. As a direct consequence, we will, hereinafter, consider our minimal viable product, MVP, as only containing proper form functionality. It should, however, be accentuated that we are not dismissing the notion of motivation entirely; indeed receiving feedback, and especially positive feedback, is a strong source of motivation on its own [34].

# Chapter 4

# Requirements Specification

This chapter concludes the work conducted throughout the previous chapter by establishing a selection of system specific requirements alongside overall, general system quality attributes we deem necessary for both the initial and future development of *FormAssist*. Indeed, whilst we opt for developing what is a relatively confined aspect of *FormAssist*—supporting proper form functionality—we wish to remain mindful of future development as to not inadvertently limit or otherwise restrict ourselves.

We have, therefore, determined a number of system quality attributes that will help steer the development and subsequent evaluation of *FormAssist*. Actively accounting for these quality attributes will aid us in developing a functioning and robust product. It is, consequently, vital that these attributes are well-established before proceeding with specifying our requirements and subsequently addressing *FormAssist*'s design and later implementation. We consider the following attributes umbrella-like terms in the sense that they encapsulate general, architectural qualities of *FormAssist* and should, therefore, not be considered specific requirements. Thus, the reader should assume each requirement is specified with these in mind where applicable.

Rather than conducting an extensive stakeholder analysis, we briefly examine the key stakeholders of this project. In this regard, we deliberately choose not to consider Mr. Jensen and Mr. Dahlberg our customers because we are not developing a product with the intention of selling it. We instead consider them co-creators because they heavily influence the inception and initial direction of the project. We, furthermore, consider the average user as someone who has a selection of prescribed exercises to perform which they have a vested interest in (a) actually remembering to perform, and (b) performing correctly. We thus consider the patients of physiotherapists as our users, and will hereinafter refer to them exclusively as users accordingly.

We have selected the following attributes as a result of internal discussions about our expectations for a candidate release of *FormAssist* in conjunction with the descriptions offered in various papers [35, 36, 37]. Please note that since there exist numerous attributes—the aforementioned sources alone list all from about a dozen to almost 50—us explicitly choosing just the following does not mean we dismiss the value other attributes might offer, rather, we consider this selection of system quality attributes as those which most accurately encapsulates how we envision *FormAssist*.

- **Extensibility** is the measure of how flexible or strict a given software product is—that is, the ease of which changes accommodating future growth and additional requirements can be implemented. In order to develop an application which not merely supports exercises and the functionality necessary to provide feedback on the performance of those exercises to its user, it is thus important that the underlying software is extensible. The exercises prescribed by physiotherapists are not necessarily a static entity—not only do different users require different treatments, which translates to *FormAssist* needing to support a wide range of exercises, it is also possible that, as the field of physiotherapy advances, new exercises may be introduced or old ones changed. The system must, therefore, support the ability to extend the repertoire of exercises. This attribute also introduces the possibility of using *FormAssist* outside the realm of physiotherapy as a general tool for improving movements. We will return to this thought in section 8.3.

- **Usability** involves how effectively end-users can use, learn, and control the system. *FormAssist* is intended to be used by people who are prescribed exercises by their physiotherapist(s). The people who will be using the system will thus, unarguably, hold varying levels of not merely technical capabilities. Yet, we must accommodate all users, and expect physiotherapists to adequately instruct users in how to install and use *FormAssist*. However, since the responsibility of actually using the application ultimately falls on the user, it is, nonetheless, important that we design *FormAssist* such that it makes accessing feedback and otherwise using the application as easy as possible. Indeed, if the user does not understand how to use the system, or utilise the provided feedback, the entire purpose of *FormAssist* ultimately falls short.

- **Portability** defines the usability of the same system in different environments. In the context of *FormAssist*, portability is chosen as an underlying priority because we want to develop an application which is accessible to as many users as possible—something which, by natural extension, means supporting both Android and iOS.

Having established the above system quality attributes, we now proceed to defining the requirements for a satisfactory MVP of *FormAssist*. In order to do this, we will solve a generic use-case of *FormAssist* and extrapolate specific requirements accordingly.

> A user has been prescribed three exercises which they have to perform daily over the course of ten days. Their physiotherapist helps them download, install, and use *FormAssist* on their Android or iOS device before they leave the clinic. The following day, the user opens *FormAssist* and navigates to the first of their three prescribed exercises as instructed by their physiotherapist. The user then selects the exercise and their screen changes and instructs them to start the exercise and otherwise position their smartphone and themselves such that *FormAssist* can see the user in full. Following a countdown in which the user has positioned themselves as instructed, the user then begins to perform the chosen exercise. As the user performs the exercise, *FormAssist* provides real-time feedback as to assure the user that they remain on the right track. Once all repetitions are completed, the user walks back to their smartphone and is provided with an overall evaluation. If any repetitions were subpar, appropriate feedback is given, otherwise positive feedback to encourage and motivate is given.

By extrapolating from this use-case, we find the following general requirements:

1. **A user must be able to select an exercise**
   A user must be able to indicate which of the supported exercises the user intends to perform in order for *FormAssist* to evaluate their performance.

2. ***FormAssist* must support multiple exercises**
   Physiotherapists can prescribe one or more exercises to be performed at home. These exercises can vary and, therefore, *FormAssist* must support a multitude of exercises. Supporting an exercise means that users can select it from an interface and that *FormAssist* has a set of heuristics for correctness evaluation. The specifics of these heuristics are wholly dependent on the exercise in question, though they can include, as previously mentioned, angles, distances, or positions.

3. ***FormAssist* must instruct the user on the positioning of the recording device**
   In order for *FormAssist* to record and evaluate an exercise being performed, the device must be placed appropriately, such that the user is in full view of the camera. *FormAssist* must be able to instruct the user on how to place the camera in a correct position.

4. ***FormAssist* must support both Android and iOS**
   Android and iOS are the most popular operating systems for mobile devices [38] and, therefore, in order to have the ability to assist as many users as possible, *FormAssist* must run on both platforms.

5. ***FormAssist* must be able to estimate a user's pose based on an image**
   In order for *FormAssist* to be able to evaluate a user's performance of a given exercise, it must, inherently, be able to estimate the user's pose during the execution of said exercise. As *FormAssist* captures the execution of a given exercise, it will receive a sequence of images of

which each image must be analysed such that the user's pose can be estimated accordingly.

6. ***FormAssist* must be able to detect repetitions of an exercise based on a sequence of images**
   For *FormAssist* to properly evaluate an exercise as per requirement 7, it must be able to detect each repetition of an exercise.

7. ***FormAssist* must be able to estimate correctness based on pose estimations**
   Requirement 6 allows the application to distinguish between repetitions. This makes it possible to evaluate each repetition individually and determine the correctness of the repetition as per the heuristics included through requirement 2, as well as the exercise as a whole.

8. ***FormAssist* must provide real-time feedback**
   We will be using real-time feedback to inform the user when they have performed a repetition correctly as per the heuristics included through requirement 2. Depending on the heuristics, real-time feedback could, for instance, be provided when a certain angle has been reached, or if a certain position is matched. This feedback must be provided in a suitable manner to avoid distracting or otherwise negatively impacting the performance of the exercise.

9. ***FormAssist* must be able to estimate a user's pose a minimum of ten times per second**
   For *FormAssist* to be able to adequately detect repetitions and provide real-time feedback, as per requirement 6, it must be able to process a sufficient number of images, or frames, each second. The lowest acceptable number is ten frames per second as we estimate that this will be sufficient in order to capture the full range of motion for an arbitrary exercise[1].

10. ***FormAssist* must provide an overall evaluation once the exercise is complete**
    *FormAssist* will, in addition to real-time feedback, provide a more detailed evaluation of the entire exercise after completion. Furthermore, *FormAssist* will also provide repetition specific feedback within the overall evaluation.

Having established our requirements, we now proceed to designing *FormAssist*.

[1] We will return to the validity of this estimation in chapter 7

# Chapter 5

# Design

In the previous chapter, we delineated the requirements we consider necessary for developing a satisfactory MVP of *FormAssist*. We, additionally, noted how we want to keep future development in mind as we specified those requirements. However, unless a MVP—which ultimately lays the foundation for any future development—is also designed with future development in mind, any such development may inadvertently be hampered or otherwise limited, no matter how well-formulated the underlying requirements may be. Correspondingly, we opt for continuing our active emphasis on ensuring a high level of extensibility, seeking to ameliorate future development. The following design considerations will thus reflect this emphasis accordingly.

Having established the essential requirements for a MVP of *FormAssist*, we can begin to compartmentalise the different requirements and establish a preliminary architecture in accordance with the use-case presented in chapter 4. We visualise this overall flow in a preliminary sequence diagram as shown in fig. 5.1. This high-level illustration of *FormAssist*'s overall functionality, moreover, establishes a number of success criteria which we can use to gauge our progress towards developing a satisfactory MVP, specifically (a) *FormAssist*'s ability to detect poses, (b) *FormAssist*'s ability to differentiate repetitions, and (c) *FormAssist*'s ability to determine the correctness of each repetition to provide both real-time and post-performance feedback.

In the following sections, we will expound the intended design for each individual module identified in fig. 5.1—that is, we will introduce its responsibility and its associated challenges. It should be noted that due to both the novelty of performing body pose detecting using only a smartphone [39], and the composite nature of this project, we will heavily rely on prototyping as a means of making informed decisions throughout the subsequent implementation. This means that the following design pages will remain fairly general as to establish, in broad strokes, what we wish *FormAssist* to include in accordance with our requirements as opposed to presenting specific frameworks or libraries.

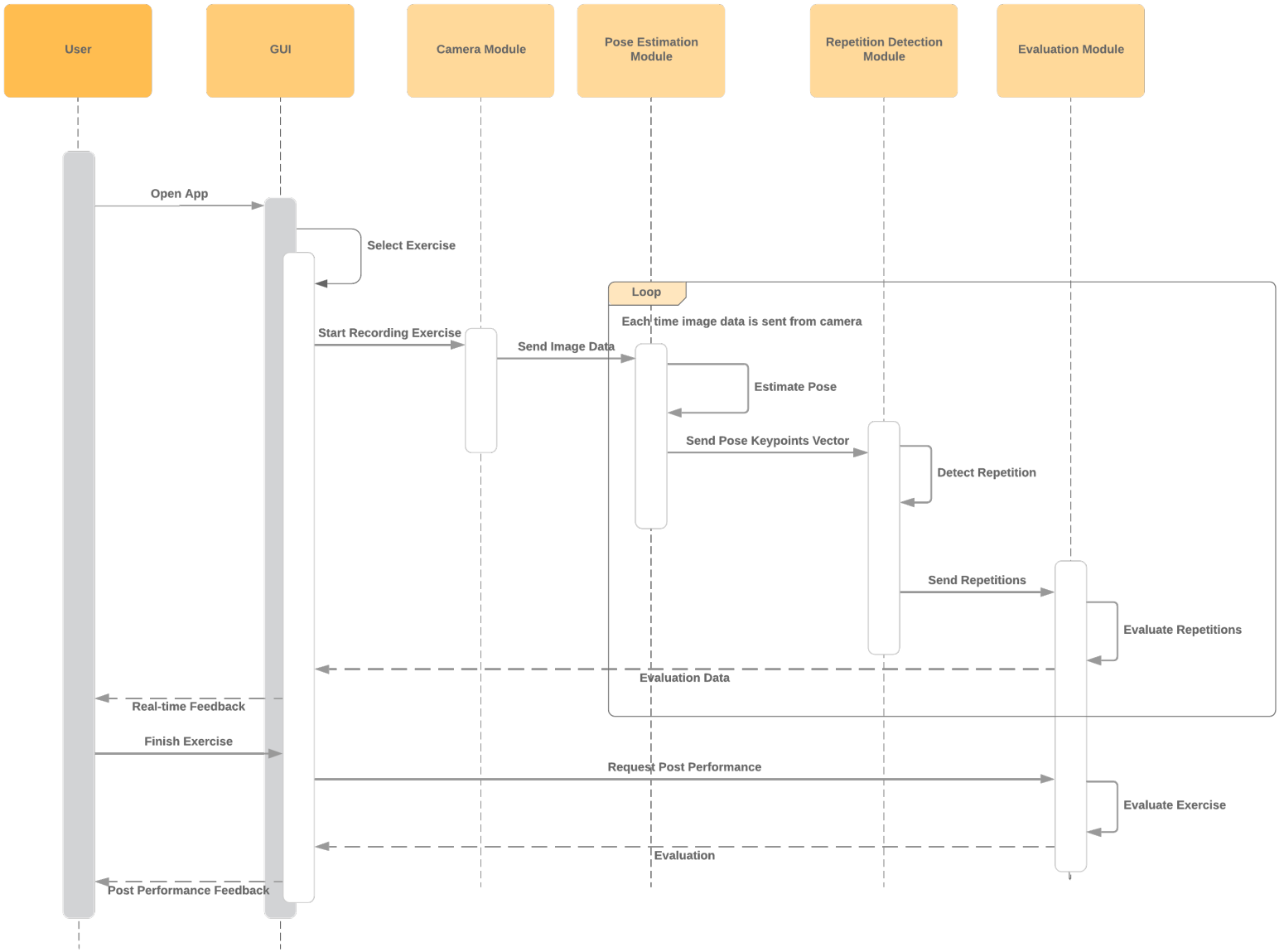



**Figure 5.1:** A preliminary overview of *FormAssist*'s flow in accordance with the use-case outlined in chapter 4.

## 5.1 Camera and Pose Estimation Modules

Once the user has started *FormAssist* and navigated to a prescribed exercise, the system moves to the two middle modules in fig. 5.1—that is, the *camera* and *pose estimation* modules. These modules are responsible for first recording a user performing an exercise and, subsequently, estimating the user's position and movement in the recording. The final output of the pose estimation module is a vector consisting of tuples representing key points with $x$ and $y$ coordinates indicating the positions of body parts. It should be accentuated that whilst we present these modules as independent modules, they are closely related to the point of effectively being a continuation of one another—indeed, the only thing which the camera module should do is to immediately supply image data to the pose estimation module, and nothing else.

### 5.1.1 Responsibility

The application will take advantage of the smartphone's camera to take images of the user performing an exercise. In the context of developing a MVP, we will only use the front facing camera. This allows us to utilise the screen to display information to the user when in use. We imagine that the reduced quality of the front facing camera compared to the camera on the back will have little to no impact on the pose estimation capabilities of *FormAssist*. The image data will be used as input for pose estimation. Some pre-processing of the image data might be required to reduce the size or to convert it into a format compatible with the pose estimation module. In this regard, we face a potential issue if too few images are processed per second as this could lead to issues with adequately detecting repetitions due to large gaps of time between image data. It is, therefore, a priority to properly test the performance of the pose estimation module to ensure an adequate frequency of frames from the camera to the pose estimation as well as the subsequent modules. We currently do not have any specific information regarding the performance, and will thus focus our efforts on prototyping multiple solutions for testing before deciding on a final design.

We briefly introduced the notion of body pose estimation in section 3.1 in connection with Microsoft's Kinect. We, additionally, noted how that in order to determine the correctness of a given exercise, whatever software solution is developed must, inherently, be able to not only capture the exercise being performed, but also be able to distinguish between the body performing the exercise and everything else. We will, consequently, investigate and present a number of such solutions and draw inspiration from them.

Before we examine existing solutions, it is pertinent to explain what body pose estimation actually is and how it works. In short, body pose estimation refers to a general problem in the field of computer vision. The essential idea is to estimate the position and orientation of an object—that is, estimate key point locations which describe the object in question. In the context of estimating a human pose, what is required to estimate and localise is essentially the major joints of the human body; e.g. shoulders, ankles, knees, elbows, and thus forth. This means that with a functioning model, one could, for instance, determine where an elbow or a knee is in an image. It should be noted that this technology is not capable of recognising *who* is in the image—it merely estimates where key body joints are [40]. A possible result of employing pose estimation to an image is shown in fig. 5.2. The

amount of key points, however, depends entirely on the employed model.

**Figure 5.2:** A possible output with highlighted key points.

During our research of existing pose estimation solutions, we found that a majority of them utilise Microsoft's Kinect, multiple cameras, or some other form of depth perception—something which is not readily available for smartphones [41]. We, therefore, cannot assume that the libraries used in said solutions immediately can be employed by us in developing *FormAssist* given the inherent limitations of most smartphone cameras. However, to broaden our understanding and otherwise increase our repertoire of possibilities, we will examine a couple of these solutions.

One solution, Pose Trainer, is an existing desktop application which, using a web camera, estimates the users' exercise pose and provides personalised, detailed feedback on how the user can improve on one of their four supported exercises [39]. Its authors note in their *future work*-section that exporting their solution to mobile devices would be a possible extension. They, additionally, note possible extensions such as providing improved feedback—i.e. through an annotated playback of the user's performance. Beyond addressing these possible extensions in *FormAssist*, we can draw inspiration from how the authors handled the various challenges they faced. Specifically, for pose estimation, they employed a pre-trained model called *OpenPose* [42]. OpenPose is a real-time multi-person system which can detect body, hand, facial, and foot key points—totalling 135 different key points—on single images. Although not necessarily needed in our case, it also supports pose estimation for multiple people in the same image.

OpenPose conveniently lists a number of alternatives on its GitHub-benchmarking section [42]. Two of the most significant competitors are AlphaPose [43] and Mask R-CNN [44], respectively. According to OpenPose's own comparison, OpenPose performs better than the two alternatives when comparing runtime proportional with the number of people per image. A fourth alternative is Facebook AI Reasearch's software system, Detectron [45], which is a library that implements various object detection algorithms. Detectron, being a wholly different alternative, differs from OpenPose

and AlphaPose in one, but significant, way: Its purpose is not to exclusively detect people, but rather any object(s) of interest—whether it be a bicycle, a car, or a pigeon. In the scope of our project, it may, therefore, not be relevant to use this library as it provides extensive functionality which goes beyond the scope of *FormAssist*. It may, however, prove relevant in future development if we wish to incorporate support for exercises which include objects.

It is not immediately possible for us to make a decision on which model to employ at this point in time. The main reason for this is that none of the identified options are specifically designed for, nor directly implemented on, smartphones. We, consequently, need to develop a number of prototypes using the various models to see whether it is possible to port it, and, if so, whether their performance is satisfactory such that we can process at least ten frames per second in accordance with requirement 9. Additionally, we ideally want to use the same framework for both Android and iOS in accordance with requirement 4 as well as to ease the overall development process.

### 5.1.2 Potential Challenges

If these modules do not work, we are effectively unable to answer our problem statement as we cannot estimate a user performing their exercise(s), nor, as a direct consequence, can we detect repetitions for evaluation. Thus, this module has a significant responsibility. We have, in this regard, identified two major points of interest when it comes to potential challenges—(1) pose estimation, and (2) performance.

Any of the abovementioned models have individual limitations on what they can estimate—i.e. the number of possible key points varies greatly depending on the model. It is, however, possible that not all key points are needed as the correctness of some exercises might only depend on a few individual key points. This, essentially, means that whilst we do require each key point, regardless of framework, to be as exact as possible, we may not require that all key points return a value in every frame. This especially becomes relevant when considering how accurate a given framework is at detecting poses from the front versus the side. Indeed, some exercises, like lunges, require *FormAssist* to view the person from the side, whereas other exercises like leg abduction should be analysed from the front. It should, once again, be noted that since we rely on smartphones exclusively, we do not have the depth perception otherwise offered by other cameras.

Another concern is the performance of the model. We will need to continuously run the model on the images received from the camera. This can cause a bottle neck if the pose detection is too slow. We will, additionally, be limited to the number of frames the model can process each second. A slow model will directly impact our ability to provide constructive, real-time feedback to the user.

## 5.2 Repetition Detection Design

Continuing our walk-through of the design of each component of *FormAssist*, we now proceed to the design of the module responsible for detecting repetitions.

### 5.2.1 Responsibility

As an exercise is being performed by the user, *FormAssist* processes each received frame as outlined in section 5.1. The outcome of these steps is ultimately a vector containing a number of key points—each key point consists of an $x$ and a $y$ coordinate—which, all together, outlines the estimated pose in the specific frame alongside a timestamp. In other words, each vector represents the identified body pose key points present in a *single* frame. The immediate challenge we face is that we have to analyse what, effectively, becomes a stream of key points without having any guarantees of how many frames per second—we receive, nor how long it takes for a user to perform a given repetition. *FormAssist* must, moreover, be able to differentiate not only individual repetitions, it must also be able to determine when the exercise has been fully completed such that post-performance feedback in accordance with requirement 10 can be provided.

Additionally, it is not enough to merely detect repetitions, real-time feedback must, in accordance with requirement 8, also be provided. However, since the smartphone in question, inherently, will be placed away from the person performing the exercise, showing something on the screen is not an ideal solution. Initially, we considered flashing the screen in green or red depending on the correctness of the detected repetition, but such a solution unarguably requires the user to continuously look at their smartphone—something which is not necessarily ideal for every type of exercise. With this in mind, we opt for audio feedback as this can be provided independently of where the user is looking and how far—to a certain extent—the user is from their smartphone. Furthermore, if the user is listening to music on the same device as *FormAssist* is running whilst exercising using a Bluetooth headset or similar, feedback can still be provided without any issues. We thus imagine that a given sound cue is played, indicating when one has reached full range of motion of the movement during a repetition where the targeted muscle is contracting—henceforth referred to as the *positive phase*. A sound cue could also be used to indicate full range of motion on the movement during a repetition where the targeted muscle is lengthening—henceforth referred to as the *negative phase*.

In accordance with requirement 10, we also want to provide more comprehensive feedback after the exercise has been performed in order to give more insightful corrections than what can be provided whilst the exercise is actively being performed. The basic idea is that we want to limit the amount of information relayed to the user whilst they are busy performing the exercise, without completely leaving them in the dark. Consider the following example: A user has to perform ten repetitions of a leg abduction exercise. This is, in short, an exercise where the user is in a standing position and has to move one leg to the side and return to the starting position, as shown in fig. 5.3.

**(a)** Start position. **(b)** Partial extension. **(c)** Complete extension.

**Figure 5.3:** Leg abduction exercise.

*FormAssist* should then alert the user when they have reached the optimal angle as their leg has extended $x$ degrees (fig. 5.3c), and when they have contracted the leg back to the approximate starting position (fig. 5.3a). This way, the user will receive two sound cues indicating that their repetition was performed correctly based on pre-defined, exercise specific heuristics. In this specific case, we are interested in preventing the user from performing the exercise incorrectly by not reaching full range of motion. Any flaws with the performance of the exercise beyond not moving the full range of motion will be left to post-evaluation. This could entail corrections such as asking the user to keep either leg straight or maintaining an upwards position i.e. not swaying their torso. Note that we have not specified a sound for when a move was incorrect. This is intentional, though it can be changed at a later point. The reason for not playing two different sound cues is twofold: (1) We do not wish to overwhelm the user with noises—something which would become a genuine problem in exercises where each repetition is fairly quick, and (2) receiving no cue is in and of itself also a cue. This design thus allows the user to only listen for a single cue and know that when they hear it, they are on the right track. Of course, every exercise eventually supported by *FormAssist* would, we expect, need specific heuristics determining correctness.

### 5.2.2 Potential Challenges

We have identified a couple of possible challenges associated with detecting repetitions. Firstly, since we want to provide feedback both in real-time and post-performance, we need to be able to evaluate a given repetition as we are receiving a steady stream of vectors whilst also storing the total frames for later evaluation. Detecting repetitions once we have a full set of data, we imagine, will not be an issue. Detecting a repetition as the exercise is being performed, on the other hand, can prove challenging. This challenge is amplified by the lack of guarantee of a steady flow of frames as previously explained.

Initially, we considered detecting repetitions using the starting position of the user and computing the difference between that and each subsequent frame in order to find the timestamp at which the user had returned to the starting position and thus had completed a repetition. The issue with this approach is that we not only inadvertently assume that the user will have a correct start position—e.g. upright, facing the camera—we also assume that the user will be performing the exercise correctly and thus return to the same position in between repetitions. However, we cannot make these assumptions. Accordingly, we plan on taking a direction-based approach to the detection of repetitions in which we check the angle or position of moving limbs, at every frame, and determine, based on the previous recorded angle or position, whether the limb is in a positive or negative phase. This way, we can look at a subset of frames and determine that once the angle changes from a positive phase to a negative phase, a repetition has been performed. This technique will also allow us to detect repetitions while the exercise is being performed as well as when the exercise is done.

We expect a number of additional challenges to arise which may or may not be specific to each exercise. Some exercises may require specific implementation details in order to properly detect their repetitions. We plan on prototyping extensively with multiple refactoring steps in order to generalise as much functionality as possible. This way we can determine what functionality can be extended to cover all possible exercises and what should be implemented for each exercise specifically. We will describe this in greater detail in section 6.4.

## 5.3 Repetition Evaluation Design

This section will describe the task of processing and evaluation the detected repetitions as well as how feedback should be provided to the user based on the conducted evaluation of their performance.

### 5.3.1 Responsibility

Evaluation of repetitions is twofold in that *FormAssist* both provides real-time feedback as the exercise is being performed, and feedback after the exercise has been performed, as described in section 5.2. Providing feedback after an exercise has been performed allows for a more detailed assessment of the user's performance. Textual feedback allows *FormAssist* to provide an overview of any identified mistakes as well as provide suggestions for correcting them. The patient can, moreover, take their time processing the feedback—something which means the amount and complexity need not be limited. The real-time evaluation, on the other hand, will consist of a sound cue when the user is performing an exercise correctly. This sound cue will be used to both indicate when the user has reached the target position of the positive phase during a repetition, and when the user has reset to the correct starting position. We do not want to overwhelm the user with too much information during the performance of the exercise as we want them to focus on the exercise. As mentioned in section 5.2, requiring that a user reads the screen of a mobile device mid-exercise is not optimal. What we instead opt to do is leave any in-depth evaluation and suggestions for after an exercise has been performed which we will refer to as *post-evaluation*.

The task of the post-evaluation portion of this module is to provide meaningful repetition-specific

feedback on the exercise. Using the leg abduction example from section 5.2, this could entail comments on whether the user reached the ideal angle during the positive phase of a repetition of the exercise or possibly exceeded it, and whether both legs remained extended throughout the exercise. The idea is to provide textual feedback which presents the issues that have been detected, and in which repetitions they were detected in. This way, the user will know what they did wrong and how often.

As explained in section 5.2, for the leg abduction exercise, *FormAssist* will be checking the angle of the legs on each frame in order to determine when a repetition has been performed in accordance with the exercise specific heuristics. Using this information, the module will be able to inform the user that the angle has reached the thresholds determining an ideal angle for the positive and negative phases. Furthermore, the values can be used to aggregate the frames into specific repetitions. Once a repetition has been detected, the repetition will be evaluated in order to determine which mistake(s), if any, have been made. This information will be saved with the repetition. When the user has finished the exercise, the module can iterate over the detected repetitions to retrieve the evaluations, aggregate them into categories, and display them to the user.

Based on the evaluation done by this module, a score will be determined. Since the score will be based on the evaluation, the responsibility of determining it will also be handled by the evaluation module. For each aspect of a repetition, a score of zero to three will be supplied based on how well the user performed. As an example, for leg abduction the evaluation module will evaluate the angle of the moving leg during the positive and negative phase as well as whether both legs remain extended during the repetition. A score will be supplied for each of these aspects such that it is possible to determine the severity of a mistake, if one is detected. After the exercise is finished the scores can be aggregated into the lowest, highest, and average score in order to provide an overview of the overall performance. The overall score of a repetition will be the lowest score achieved during that repetition. This is because achieving a low score in one aspect may make a higher score in another irrelevant. In other words, the overall score of a repetition should be capped by the aspect which was scored the lowest. The user can then inspect the specific ratings and feedback for specific rating if they wish to.

### 5.3.2 Potential Challenges

One of the possible challenges we may encounter with this component is the need for very specific suggestions to improvements on exercises. These suggestions will need to be implemented on the specific exercise and will, therefore, limit the ability to create generic functionality which can be applied across multiple exercises. This component will, thus, require a larger amount of time and code to implement. This will result in a slower process when attempting to add additional exercises. Throughout the development of this component, we will therefore continuously evaluate the implementation in order to determine which functionality can be generalised to cover all exercises and what functionality needs to be implemented for an exercise specifically.

## 5.4 Testing Plan

Since *FormAssist* will consist of a number of different, interweaved components, we need to carefully plan our testing of *FormAssist* as a whole. Indeed, at its highest level, we have identified three broad categories—each featuring a different set of testing requirements—into which we can allot our different modules and their internal components. These are (1) everything related to the pre-trained pose estimation model and any associated frameworks and libraries, (2) all logic enabling repetition detection and subsequent evaluation, and (3) the elements of the system with which the user actually interacts. The following subsections will expound each identified category and its associated testing requirements. It is important to accentuate that whilst requirements based testing is a well-established process [46], we will not strictly adhere to it, but merely determine which functional and non-functional attributes need to be tested based on the requirements extrapolated from our generic use-case presented in chapter 4. Please, moreover, note that all subsequent implementation and testing will be conducted using an OnePlus 5T for Android, and an iPhone 6 and an iPhone X for iOS.

### 5.4.1 Pose Estimation Module

The first category of components can collectively be considered deterministic as its outcome primarily relies on a neural network, which is, in itself, deterministic, meaning that it will always give the same output for the same input. In essence, this means that, when configured correctly, providing the same image to these components will always output the exact same key points regardless of platform. It is, therefore, our goal to find a pre-trained model with a high accuracy. We will, additionally, prioritise well-established frameworks and libraries as these are the most likely to have already undergone extensive testing and / or feature comprehensive testing suites. This ultimately means that we need not necessarily prioritise painstakingly testing these components, but rather confirm that we receive the expected output given a specific input. We argue that we do not need to concern ourselves with testing the internal components of the chosen frameworks and libraries, but rather focus on testing the input and output with a blackbox integration test. Some models feature testing data which we can conveniently use to verify our configuration. If this is not the case, we can test the models on a single image which we manually verify. As a direct consequence, we will henceforth, and in the context of developing a MVP, focus our testing efforts on the two remaining broad categories of components

once we have confirmed the output of this module. Indeed, any subsequent development, and thus testing, will inherently require this module to function as expected.

### 5.4.2 Repetition Detection and Evaluation Module

The second category concerns the logic involved in detecting repetitions and evaluating each detected repetition—that is, the functionality which relies on the output from our pose estimation module. The majority of the functionality in this category will, therefore, largely consist of computations of various angles and body positions based on the output. These computations will require unit tests in order to ensure correctness. We, additionally, want to conduct some integration tests such that we can verify the correctness of both modules. This could, for instance, be done by processing a sequence of images with a known number of repetitions to attain a sequence of key points on which we detect repetitions for subsequent evaluation. This method can then be repeated as many times as needed on varying sizes of data in order to thoroughly verify the correctness of the components in this category. Once we have confirmed that the repetition detection component works as intended, we can employ the same method of processing a video of an exercise with known mistakes to test the evaluation component. Of course, testing the evaluation component inherently both requires that the pose estimation module functions correctly, and that the detection component is able to correctly identify repetitions.

### 5.4.3 User Interaction

The third category relates entirely to the elements with which the user interacts—that is, the graphical user interface of *FormAssist*. This is not something which we have previously discussed beyond merely requiring that the user is able to somehow select an exercise. As we are not designers, and have little experience with creating user interfaces overall, we will consult with interaction design graduate students during our implementation. We, moreover, acknowledge the importance of having a solid user interface—as also reflected by our usability requirement—and we will, consequently, conduct usability tests to gauge our progress in this regard.

# Chapter 6

# Implementation

Having established a preliminary architecture, and having identified the overall modules of *FormAssist* in the previous chapter, we now proceed to divide the group into three smaller teams and assign each to a specific step of *FormAssist*'s development. Two teams will focus exclusively on prototyping cross-platform frameworks before advancing to addressing pose estimation. The third team will initiate work on repetition detection and evaluation modules. We, additionally, assign manpower where possible to designing the graphical user interface (GUI) and its implementation. This chapter will delineate the work completed by each internal team and conclude by presenting a final system overview. We have included a simplified, preliminary system overview in fig. 6.1 based on the modules identified throughout chapter 5.

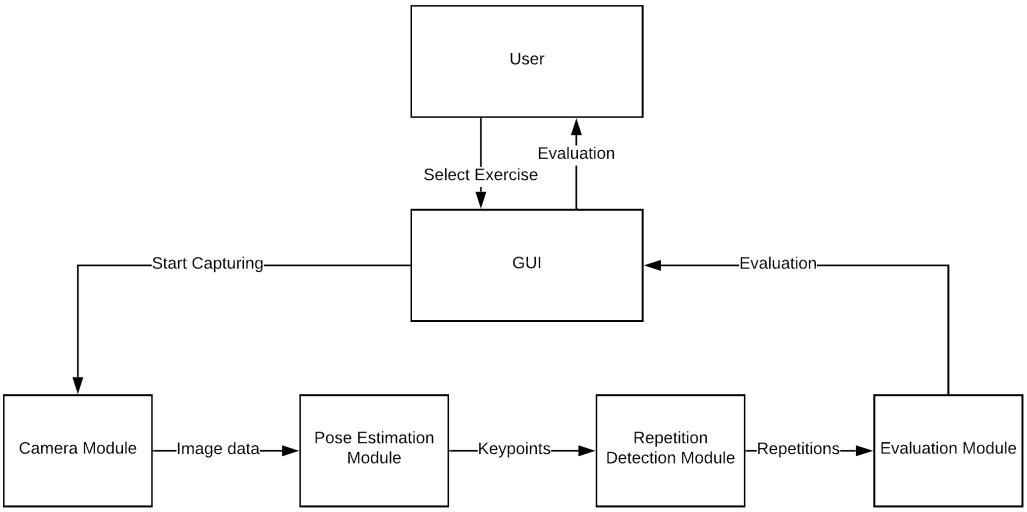



**Figure 6.1:** Preliminary system overview.

## 6.1 Framework

As established in our section 3.2, and in accordance with our focus on portability alongside requirement 4, we want to support both Android and iOS. We thus, ideally, need a framework which supports cross-platform development. The alternative is developing and subsequently maintaining what effectively becomes two applications for two operating systems—something which is both a tedious and potentially unnecessary task. We, consequently, want to minimise native development as much as possible to ease our development process. We are not the only ones who have faced this issue, and numerous efforts towards supporting cross-platform development have thus been taken accordingly. The first attempt at building a cross-platform framework relied on web views using HTML5 [47]. Whilst HTML5 paved the way for cross-platform development, it also introduced an extra layer between the application and platform which, inevitably, caused an overhead and ultimately reduced performance compared to that of native applications. In order to address this issue, newer frameworks were subsequently developed. We will examine the three most popular ones: Flutter, React Native, and Xamarin [48].

### 6.1.1 Flutter

Flutter is a free, open-source framework for developing cross-platform applications developed by Google [49]. Flutter is based on the programming language Dart [50], and features its own, high-performance rendering engine written in C++. Dart uses ahead-of-time compilation which, in this case, effectively translates to compiling into native, system-dependent machine code such that the application can be executed natively and thus achieve performance similar to that of native applications [51]. Flutter was initially released in 2017 for developers, and has since received numerous updates with the most recent major update, at the time of writing, being the 1.0 release in December 2018. Flutter, moreover, features a strong community with a fast-growing ecosystem. However, given its age, there is a legitimate risk that resources, documentation, and libraries for certain functionality may be scarce. It is, conclusively, worth noting that Flutter applications can be debugged in both a virtual environment using emulators, or directly on a device, and allows for state based hot reload, which means that the state is kept when functionality is added or modified during runtime [52].

### 6.1.2 React Native

React Native is similarly a free, open-source framework for developing cross-platform applications, though React Native relies on JavaScript and is developed by Facebook [53]. More specifically, React Native is based on React [54], which is a JavaScript framework for building user interfaces on the web. React Native is, at the time of writing, the second-most starred project available on GitHub [55]. Applications written in React Native invoke the native rendering APIs of Android and iOS through a designated React Native Bridge [56]. Figure 6.2 shows how communication is done between the JavaScript code and the native platform specific code through the React Native Bridge. This, inherently, induces a small overhead, but can outperform that of the HTML5 solution. React Native supports hot reloading as well as debugging through a web browser. The framework, being

released in 2015, additionally features a mature and well-supported ecosystems with an abundance of packages [57].

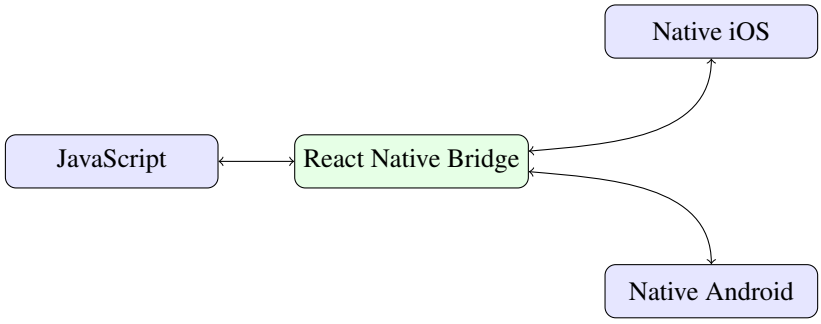



**Figure 6.2:** Communication over the React Native Bridge. Each side of the bridge can send and receive information.

### 6.1.3 Xamarin

Xamarin is currently maintained by Microsoft [58], and is the third cross-platform framework we consider. The technology as a whole consists of three core components—(1) Xamarin Platform, (2) Xamarin Cloud, and (3) Xamarin Insights—all of which are written in C# and use native libraries wrapped in a .NET layer [59]. Xamarin features native API access and contains standard, native user interface controls. Thus, applications developed in Xamarin can match the performance of native applications. However, achieving that performance does require some native development—e.g. to develop the UI using *Xamarin.Forms* [60]—which arguably goes against the notion of cross-platform development. Additionally, it should be noted that Xamarin does not provide hot reloading, which can make debugging tedious.

## 6.2 Choosing a framework

One matter is choosing a framework which enables us to develop for both Android and iOS simultaneously, another matter entirely is choosing a framework which adequately supports pose estimation functionality—that is, supports machine learning libraries. The previously introduced models—OpenPose, AlphaPose, and Mask R-CNN—are all pre-trained models and can thus be considered plug-n-play-like solutions. Our challenge is employing them on a smartphone—that is, using model inference on a smartphone through a smartphone compatible library.

We relatively quickly dismiss Xamarin as a result of how Xamarin does not offer hot reloading, and because of the previous experiences with the framework from one of the group members. Additionally, Xamarin has the least amount of users and libraries of the three possibilities explored. Whilst there exists some libraries for .NET which support TensorFlow, and even other machine learning libraries like *Infer.NET* and *ONNX*, e.g. *ML.NET* [61], or libraries which offer bindings to TensorFlow, e.g *TensorFlowSharp* [62], very few, if any, .NET machine learning libraries are available for mobile platforms. Those which do advertise Android or iOS support, e.g. *TensorFlow.Xamarin.Android*

[63], do not appear to be actively developed. We, thus, continue examining Flutter and React Native.

In machine learning, it is common to make distinction between *learning* and *inference*. The former is associated with estimating parameters, whereas the latter is considered making a prediction (e.g. with linear regression one is able to predict some real valued variable given a set of features and learned parameters). Indeed, training, or learning, happens first, and once a model is adequately trained, it is applied to new data—i.e. our user performing an exercise—and information is inferred accordingly [64]. It is in this regard important to note that since we are not training any models, but merely applying a pre-trained one to conduct inference, our ability to estimate poses will remain static and not improve unless we employ a new model. The reason why it is not possible to improve is that we are unable to gather ground truths to train with—i.e. data for the exact location of each body part that we wish to estimate. We will, however, in accordance with our extensibility requirement, be prioritising an implementation which does not restrict the possibility of upgrading the chosen model at a later date.

The most important factor in choosing either Flutter or React Native over the other is thus arguably the ease of which we can implement pose estimation using the given framework. We are working on the cutting edge of what is possible in terms of model inference on mobile [65, 66, 67], and as such, there are only a few examples of cross-platform applications using model inference currently available. Similarly, the few available libraries which do offer such functionality are still in early stages of development, supporting only a confined set of models. In the case of Flutter, the ones we found only support models for object recognition—e.g. distinguishing and labelling various common objects such as loud speakers, cutlery, and plants—and not pose estimation [68]. Additionally, those available for React Native, whilst more mature, only supported facial, textual, and barcode recognition [69]. Thus, in both cases, neither can be considered applicable to our use-case. It is, therefore, up to us to develop the necessary functionality to support cross-platform inference for pose estimation. We have two options, namely (1) extending existing libraries for either Flutter or React Native with the functionality which is currently missing, or (2) developing our own library doing exactly what we need—that is, access and start the camera, and subsequently apply model inference—from scratch. Since using the camera is a very common use-case, we argue that both Flutter and React Native will support this seamlessly. The deciding factor will, consequently, be how well model inference performs using either framework.

In order to properly compare Flutter and React Native, we opt for implementing the same machine learning library using both frameworks. Conducting inference still requires significant computational power using most models, so models created specifically with smartphones in mind are necessary to reach our performance target of ten frames per second. We managed to find a couple of different models which meet this criteria; (1) Edvard Hua's *PoseEstimationForMobile*-model [70], and (2) Fritzlabs' Fritz pose estimation model [71]. The output of each model for each processed frame is 14 heatmaps, where each heatmap corresponds to one of 14 key points (e.g. head, right knee, left foot, right hand, etc.) in which redder shading represents a higher probability of the key point being in this position. An example heatmap for the head-key point can be seen in fig. 6.3. The reader may find the complete list of key points in appendix D.

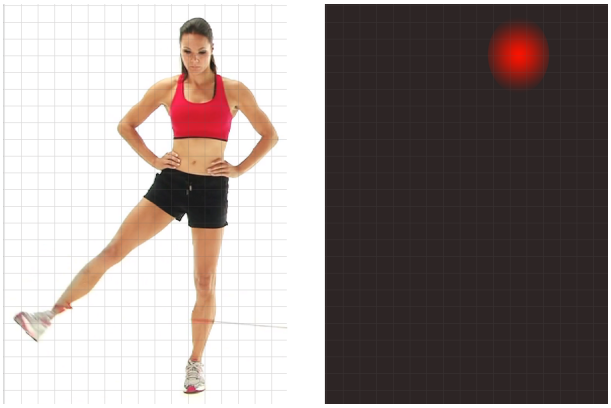


**Figure 6.3:** A woman performing leg abduction and the resulting head-key point heatmap generated by the model.

Whilst we will be working with the abovementioned models, it is relevant to note that since most machine learning libraries are built on similar basic structures, it is possible to convert models from one library to that of another [72, 73, 74]. We, therefore, opt for starting with one of the most widely adopted libraries for machine learning: Google's TensorFlow [75]. In 2017, Google launched a dedicated cross-platform mobile version called TensorFlow Lite (TFLite) [76]. Due to the very well developed ecosystem surrounding TensorFlow, and by extension, TFLite, we opt for proceeding with TFLite as our initial library. Two of our three internal teams proceed to implement pose estimation using Flutter and React Native, respectively.

### Flutter

One team quickly progressed with Flutter due to its native camera access, making it possible to subscribe to a stream of frames from live video recordings—that is, the ability to contemporaneously apply functions on each individual frame from the video. The process of using TFLite with Flutter is a bit more complicated as an official API for Dart, the programming language used for writing Flutter, does not exist. This, in effect, means that we cannot call TFLite API directly from Dart, and that platform-specific solutions are necessary [77]. A library using this approach exists, but, as previously mentioned, it only works with two pre-trained models designed for object detection. We wanted the possibility to easily trial different models. Existing examples employ a hardcoded model, making them time-consuming to modify to accommodate alternative models. Thus, we devoted time to develop our own generic implementation allowing us to swap models with ease without requiring changes to the codebase. The next step is pre-processing frames such that they can be inputted to the current model for inference. This step proved difficult, as few Dart libraries, at the time of writing, support this kind of image conversion.

At this point, we paused to evaluate Flutter and Dart as we had, unexpectedly, encountered two issues in just two implementation steps. Whilst we are confident that we could have handled the image conversion issue, internally discussing which additional steps in the implementation had to be taken, alongside searches for Flutter libraries which would support such functionality, revealed just how new and immature Flutter still is. With this, and the novelty and associated complexities

involved in developing *FormAssist* in mind, we thus decided to halt future work on this front and merge this team with the React Native team, who had made promising progress during the same period of time.

### React Native

React Native required some additional steps to configure properly such that debugging could be conducted efficiently. Despite the abundance of libraries and documentations compared to Flutter, the React Native team thus still had a somewhat slow start. It is, additionally, not possible to interact directly with the camera in React Native, as its cross-platform code is written in JavaScript and run by an interpreter in native applications for Android and iOS. The communication between the React Native code and the native system is thus done through the React Native Bridge. However, the RNB is designed for transferring event data such as button clicks, tabs and similar—it is, in other words, not designed for, handling large amounts of data like the stream of images *FormAssist* requires [78]. We, therefore, stress-tested the RNB to determine, exactly, what it can and cannot handle. We conducted two tests; one with a lot of small messages, and one sending fewer, but larger messages. In the first scenario, we sent approximately 200 messages of a few bytes each per second. This immediately hampered the performance of the application, and we only recorded one in approximately every 80 messages as arriving at the React Native side. The second test involved sending ten messages per second of size 355 kilobytes. This, too, caused a significant reduction in performance of the application with just two in ten messages arriving on average.

Despite the performance issues with React Native, the maturity and ecosystem vastly outweighs the performance related downsides. Additionally, being able to *actually* run and test various pose estimation libraries makes this a clear candidate for future implementation. We thus continue our implementation using React Native.

## 6.3 React Native Implementation

This section delineates the incremental process of identifying and prototyping a number of models for pose estimation following our preliminary research outlined in chapter 5. To briefly summarise the two-fold responsibility of this module, we first need to access the front facing camera of the smartphone and record video. As we record, we simultaneously need to run inference to estimate poses on at least ten frames, i.e. images, per second to enable subsequent repetition detection and evaluation by other modules. Please note that we, hereinafter, will be referring to the front facing camera simply as camera. This means that the preliminary system overview now looks as seen in fig. 6.4

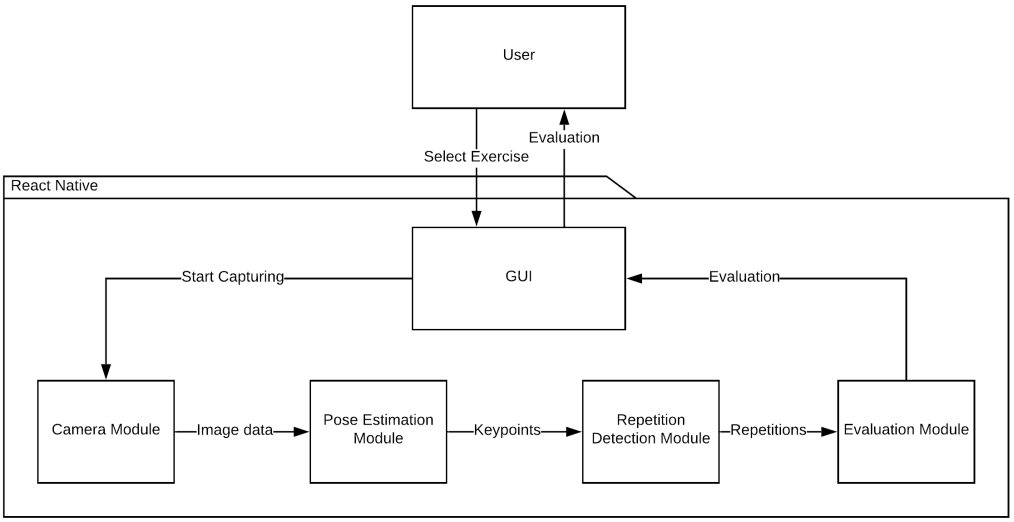



**Figure 6.4:** Updated preliminary system overview.

As noted in the previous section, it is not immediately possible to interact with the camera on neither Android nor iOS directly through React Native. We thus refer to the library aptly named *React-Native-Camera* (RNC) [69] which controls the smartphone's camera on the native side, but allows for API calls through the RNB such that cross-platform development using React Native remains possible. It, additionally, features a comprehensive set of mappings for common camera related events—e.g. zooming, focusing, and similar triggers—for both Android and iOS. None of the supported events include any image data, which instead is displayed directly on the screen by native code. Considering how RNC is both the only existing library, and how it already features a comprehensive array of functionality, we opt for extending RNC to accommodate our machine learning requirements as opposed to developing a library from scratch. However, given that we are using the React Native framework, we are inherently restricted by the limitations of the RNB. This effectively means that we need to handle everything related to the camera and pose estimation on the native side as opposed to modifying RNC to send image data through the RNB. This, in effect, means that what we need to do is to extend the RNC with a new event supporting our pose estimation model. We,

therefore, need to work in Java for Android, and Swift and Objective C for iOS. Please note that for sake of simplicity, we will only showcase Java code in the subsequent code snippets.

We did not immediately have a solution for how to extend the RNC and, consequently, searched for existing projects which apply TensorFlow natively as inspiration for our own implementation. Google's TensorFlow team conveniently provides examples for on-device inference using Android [79] and iOS [80], which we can use to get started. As noted in the previous section, we have identified two models—Edvard Hua's *PoseEstimatorForMobile* [70], and Fritzlabs' Fritz models [71]—which both meet our criteria of being able to run on smartphones with ten images per second. We thus proceed, seeking to apply both models in accordance with the existing TensorFlow examples. This entails loading the model-file and specifying the input and output sizes accordingly—that is, we need to call a setup function in which the dimensions of the nested input and output arrays are specified such that they match the specific model's requirements. In our case, for the input, that specifically means specifying how both models expect a three dimensional $192 \times 192 \times 3$ matrix, where $192 \times 192$ is the width and height of the image, and 3 represents the colour values red, green, and blue. The output of both models is also a three dimensional matrix, though with dimensions $96 \times 96 \times 14$, where $96 \times 96$ represents the number of pixels in the input image scaled down—this specific number is a result of the internal construction of the chosen models, and other models may, thus, have different dimensions. 14 corresponds to the number of heatmaps generated by the model. That is, we have one heatmap for each estimated body part. This process is visually depicted in fig. 6.5, following the flow from the input image until the heatmaps are generated. The remaining parts of the diagram will be elaborated upon later in this section.

We need to be able to initiate our model with the output of the camera. Since both models require images with size $192 \times 192$ and three colour values per pixel, we configure the function shown in listing 6.1 to prepare for the subsequent pose estimation.

```
1  private static final int DIM_BATCH_SIZE = 1;
2  private static final int DIM_PIXEL_SIZE = 3;
3
4  PoseEstimator(Activity activity) throws IOException {
5      tfliteModel = loadModelFile(activity);
6      tflite = new Interpreter(tfliteModel, tfliteOptions);
7      imgData =
8          ByteBuffer.allocateDirect(
9              DIM_BATCH_SIZE
10                 * getImageSizeX()
11                 * getImageSizeY()
12                 * DIM_PIXEL_SIZE
13                 * getNumBytesPerChannel());
14
15     imgData.order(ByteOrder.nativeOrder());
16 }
```

**Listing 6.1:** Java code for creating an instance of the TensorFlow PoseEstimator.

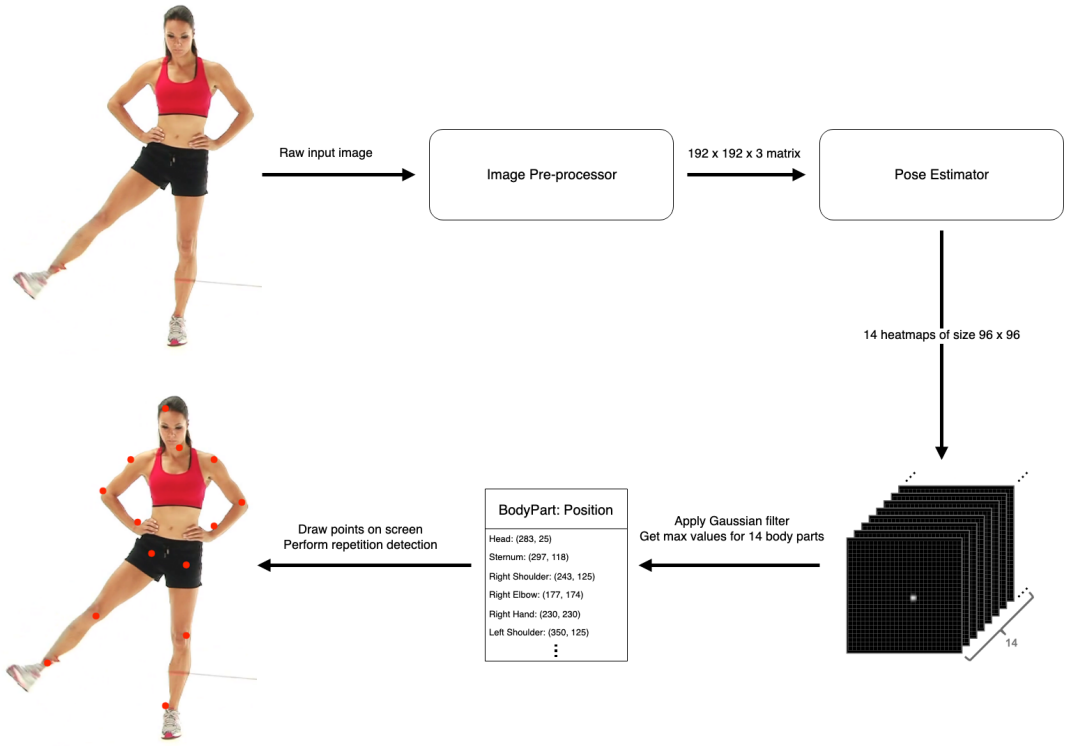



**Figure 6.5:** The process of transforming input frames into body part positions.

The snippet in listing 6.1 ensures that we have an instance of the `PoseEstimator` running by loading the model and configuring the input and output sizes, as seen in lines 5-13. We can override the existing `onFramePreview` function offered by RNC, as shown in listing 6.2. This function is intended to allow for the application of functions on each individual frame—i.e. exactly what we want. It is in this regard important to accentuate that we, consequently, do not store the image data on the device, we merely opt for processing the preview frames.

```
1  private static final int IN_DIM = 192;
2
3  @Override
4  public void onFramePreview(
5      CameraView cameraView, byte[] data, int width, int height, int rotation){
6
7      // The input image is in YUV format, we need RGB
8      int[] bytesInRGB = AndroidImageUtilities.convertYUV420_NV21toRGB8888(data, width, height);
9
10     // Create a bitmap from the byte array
11     Bitmap largeBitmap = Bitmap.createBitmap(
12         bytesInRGB, width, height, Bitmap.Config.ARGB_8888);
13
14     // Scale down the bitmap to 192 x 192
15     Bitmap scaledBitmap = Bitmap.createScaledBitmap(
16         largeBitmap, IN_DIM, IN_DIM, true);
17
18     poseEstimator.convertBitmapToByteBuffer(scaledBitmap);
19     poseEstimator.estimatePoseInBackground();
20 }
```

**Listing 6.2:** Java code for creating and scaling a bitmap before running inference for pose estimation.

The process of converting to $192 \times 192$, as required by the model, proved more cumbersome than expected due to (a) column-major versus row-major [81], and (b) RGB (red-green-blue) versus BGR (blue-green-red) colour models. The former being an issue because the frame received from the camera regardless of platform is a byte buffer—that is, one long buffer which needs to be indexed correctly. The latter being relevant because the individual colour values are put into an array, and some models expect red to be the first value, and others blue. There is, additionally, no general consensus as to which choice of either column-major versus row-major should be used when developing models, and the documentation for our specific models did not elaborate on this topic. This is also the case for RGB vs BGR—that is, the order of the values representing each of the three colours.

The format of the input byte array `data` is in the YUV colour encoding format [82], and thus needs to be converted to RGB before we can create a bitmap which we need for inference. We employ the pre-defined `convertYUV420_NV21toRGB8888` function to do this, as seen in line 8 in listing 6.2. We then generate a bitmap, as seen in line 11. The reason why we need to go through this step as opposed to scaling the bitmap immediately, as we do in line 15, is because we want the entire image scaled and not merely cropped. Once the bitmap has the desired dimensions, we convert it once again to a byte buffer as this is the format which the model expects the image to be in. This is done in line 18. We then need to run inference—that is, actually apply our model—to the resulting byte buffer.

We use the function shown in listing 6.3, to handle the conversion.

```
1  /** Writes Image data into a {@code ByteBuffer}. */
2  private void convertBitmapToByteBuffer(Bitmap bitmap) {
3      if (imgData == null) {
4          return;
5      }
6
7      imgData.rewind();
8      bitmap.getPixels(
9          intValues,
10         0,
11         bitmap.getWidth(),
12         0,
13         0,
14         bitmap.getWidth(),
15         bitmap.getHeight()
16     );
17
18     // Convert the image to floating point.
19     int pixel = 0;
20     for (int i = 0; i < getImageSizeX(); ++i) {
21         for (int j = 0; j < getImageSizeY(); ++j) {
22             final int val = intValues[pixel++];
23             addPixelValue(val);
24         }
25     }
26 }
```

**Listing 6.3:** Java code for converting a bitmap to a byte buffer.

Notice how `convertBitmapToByteBuffer` modifies the `PoseEstimator` property `imgData`. This is used as input to the model. Once correctly converted, we call `estimatePoseInBackground` to run inference, as seen on line 19 in listing 6.2.

We now have a partial implementation using TFLite which allows us to access the camera and apply the model for inference. We decide to measure the conversion and inference times at this point to determine whether we are on the right track. This implementation proves to be too slow for our requirement of ten frames per second on both Android and iOS. It is too slow for two reasons: (1) The TFLite inference times proves to be around 150ms per frame on an OnePlus 5T, 130ms on an iPhone X, and 500ms on an iPhone 6—this translates to a framerate of approximately 6 for the OnePlus 5T, 7-8 for the iPhone X, and approximately 2 for the iPhone 6, and (2) the image preprocessing contains unnecessary and time-consuming conversions, which also contributes to the poor performance. Indeed, if too few frames are processed per second, it becomes infeasible to do repetition detection and subsequent evaluation. This is because a lack of frames introduces the risk of missing key parts of the exercise such as when reaching full range of motion in the negative and positive phases. The conversion code originates from the code examples supplied by Google, which happens to be developed with single image inference in mind. Consequently, the conversion code does not accommodate our need for handling a continuous stream of images.

We, therefore, backtrack slightly, seeking to develop a faster solution using another framework. Since the current results are the product of TFLite, we shelve TensorFlow for now and experiment with other libraries. The reason for not changing the models is twofold: (1) We do not have a lot of models available, and, more importantly, (2) we have confirmation that at least one of our chosen models, Edvard Hua's [70], performs with an inference time at 30ms on a OnePlus 5T using the Android-specific machine learning library MACE [83].

In the spirit of keeping our code as cross-platform as possible, we decide to repeat the above steps such that we can apply the model and measure its performance; only this time using the cross-platform Fritz library [71]. Indeed, a lot of the code from the previous implementation can be reused seeing how we need to take similar configuration steps. Unfortunately, the inference time with Fritz is very similar to that of TFLite and, consequently, not good enough either. Our only remaining option is thus to employ platform-specific solutions. We, therefore, use Edvard Hua's model moving forward. It is in this regard important to accentuate that whilst we would have preferred a cross-platform solution to ease the workload, we are still adhering to our requirement of supporting both Android and iOS. In the case of iOS, we have an obvious option with Apple's own Core ML. In the case of Android, we have a couple of different options; specifically MACE and a beta version of TFLite which supports using the GPU for inference, as opposed to only the CPU, which has been the case thus far. The Core ML solution proved to be simpler as the Core ML Framework handled the image conversions necessary efficiently, as one would expect given that Apple are able to properly leverage both the software and hardware. Core ML performs at approximately ten frames per second on an iPhone 6 and about 40 frames per second on an iPhone X. The steps required to connect the output of the model to the React Native part, and therein the repetition detection module, will be explained during the Android section as they are virtually identical for both platforms.

In the case of applying the model and running inference on Android, we start by testing MACE, which is a highly optimised machine learning library built in C++. The fact that it is built in C++ is relevant because it arguably complicates the build process. MACE has a very specific set of requirements for versions and positions of libraries [84]. The creators, therefore, provide a Docker image [85] to help ease the process of building the framework. This, however, is not a complete solution because building MACE also relies on dependencies on the host machine itself. Additionally, some paths in MACE happen to be hardcoded in the codebase—something which introduces several additional complications. Eventually the example application [86] provided by MACE is built, and we arrive at results of 30 frames per second. Yet, despite being able to run the demo application, we quickly encounter a number of inconsistency issues keeping things running. One of these issues is a `SIGARBT` error that occurs on runtime. We have been unable to isolate the cause of the error. We have, therefore, been unable to integrate MACE with RNC consistently. We, additionally, encountered the error in the MACE demo occasionally. We attempted to contact the creators of MACE, but we have not yet received a reply at the time of writing. We, therefore, do not consider MACE a viable option for us at the moment, though it is very possible that MACE can be employed in future versions of *FormAssist*. We have already tested TFLite for inference, but only the most recent stable version. Google's TensorFlow team has a nightly beta build of TFLite. This version includes useful functionality such as support for inference using the GPU, which will potentially improve performance as we thus far have relied on the CPU.

Since we have not yet achieved adequate performance on Android, this becomes our main focus as we begin to implement *FormAssist* with TFLite beta. Accordingly, we create an additional layer of abstraction to ease the process of running the pose estimation via the class `PoseEstimatorRunner`, which handles running the inference on a separate background thread. Using a separate thread for heavy computations is a common practice which allows the application to remain responsive. The `PoseEstimatorRunner` creates an instance of the `PoseEstimator`-class, thereby reusing the code shown in listing 6.1. The responsibility of `PoseEstimatorRunner` is to continuously (a) fetch a frame, (b) run inference for pose estimation on the frame, (c) process the output from the model, and (d) send the processed output to React Native for subsequent processing. As mentioned, this chain of operations is handled on a dedicated thread which we start as shown in listing 6.4.

```
1  public void startPoseEstimationInBackground(){
2      startBackgroundThread();
3      backgroundHandler.post(() -> poseEstimator.useGpu());
4  }
5
6  /** Starts a background thread and its {@link Handler}. */
7  private void startBackgroundThread() {
8      backgroundThread = new HandlerThread(HANDLE_THREAD_NAME);
9      backgroundThread.start();
10     backgroundHandler = new Handler(backgroundThread.getLooper());
11
12     synchronized (lock) {
13         runPoseEstimator = true;
14     }
15     backgroundHandler.post(periodicallyEstimatePose);
16 }
```

**Listing 6.4:** Java code for starting the pose estimation module on a background thread.

Note how the GPU is attached in line 3. This functionality was not yet available in the first version of TFLite we tried, which relied exclusively on the CPU power. Once the `backgroundThread` has been started in line 9, we can control what that specific thread should run. Once the thread has been started, we post the `periodicallyEstimatePose` to the thread. Due to the recursive post call seen in line 10 in listing 6.5, we effectively run the pose estimation continuously. Note how we employ a synchronised lock in line 5: This is done to ensure that only a single instance of `estimatePose` is ever running on the thread. This loop continues until `backgroundThread` is either stopped explicitly, or when *FormAssist* is closed.

```
private Runnable periodicallyEstimatePose =
    new Runnable() {
        @Override
        public void run() {
            synchronized (lock) {
                if (runPoseEstimator) {
                    estimatePose();
                }
            }
            backgroundHandler.post(periodicallyEstimatePose);
        }
};
```

**Listing 6.5:** Java code for the runnable periodicallyEstimatePose.

The implemenation of `estimatePose` can be seen in listing 6.6.

```
private void estimatePose() {
    if (poseEstimator == null || textureView == null) {
        return;
    }

    // Grab the bitmap and scale to size 192 x 192.
    Bitmap bitmap = textureView.getBitmap(
        poseEstimator.getImageSizeX(),
        poseEstimator.getImageSizeY()
    );

    // The bitmap is null just before the user leaves the video view
    if (bitmap == null) {
        return;
    }

    poseEstimator.estimatePose(bitmap);
    sendPoseEstimatedEvent(new BodyPoints(poseEstimator.getHeatmap()));
    bitmap.recycle();
}
```

**Listing 6.6:** Java code for pose estimation on each received frame from the preview stream.

The image data, or preview frame, is retrieved using `textureView` which is a component defined directly in Android [87]. This component is used to render content such as images or 3D models in Android applications, and using it significantly speeds up the process of converting image data. Recall how we initially converted a byte buffer to a large bitmap—that is, the original size of the image—which we then scaled to a smaller bitmap before yet again processing it to match the required input of the model as shown in listing 6.3. We can, effectively, skip all these steps by utilising `textureView`, which has a function which returns a scaled bitmap. These steps are replaced by lines 7-9 in listing 6.6. However, at this point, we still need to convert the scaled bitmap to a byte buffer as before. The `PoseEstimatorRunner` contains an instance of our `PoseEstimator` classifier. The

next step is to run the classifier with the scaled bitmap data as input.

The output of running inference is the aforementioned $96 \times 96 \times 14$ three dimensional matrix. Whilst we could return this output immediately, doing so would put unnecessary strain on the RNB which could lead to a host of other issues given its inherent limitations. Additionally, what we are really interested in from this output are the 14 key points which correspond to the $x$ and $y$ coordinates of the estimated 14 major joints of the pose. We, consequently, proceed to natively convert the $96 \times 96 \times 14$ output to a vector consisting of tuples with estimated $x$ and $y$ coordinates for each the 14 key points. The conversion for Android is shown in listing 6.7. The code for iOS does exactly the same, only written in Swift. This matches what is depicted in fig. 6.5 from the heatmaps and onward in the flow.

```
1  private static final int BODYPART_COUNT = 14;
2  private static final int ROW_COUNT = 96;
3  private static final int COL_COUNT = 96;
4  private static final int COORDINATE_COUNT = 2;
5
6  private static int[][] getMaxValues(float[][][] heatmap) {
7      int[][] arr = new int[BODYPART_COUNT][COORDINATE_COUNT];
8      for (int bodypart = 0; bodypart < BODYPART_COUNT; ++bodypart) {
9          // Resetting max for each bodypart.
10         float max = 0;
11
12         // Default values bodypoint-not-found. iOS does the same.
13         arr[bodypart][0] = -1;
14         arr[bodypart][1] = -1;
15
16         // First and last rows and columns are ignored to avoid issues when blurring.
17         for (int row= 1; row < ROW_COUNT - 1; ++row) {
18             for (int col = 1; col < COL_COUNT - 1; ++col) {
19                 float gaussianValue = getGaussian(heatmap, bodypart, row, col);
20                 // If this value is the best so far, we save it.
21                 if (gaussianValue > max) {
22                     max = gaussianValue;
23                     // Flipped "back" on purpose, as the model flips the output.
24                     arr[bodypart][0] = col;
25                     arr[bodypart][1] = row;
26                 }
27             }
28         }
29     }
30     return arr;
31 }
```

**Listing 6.7:** Java code for converting the heatmaps to a vector consisting of tuples.

`getMaxValues` iterates through each pixel of each of the 14 heatmaps and computes the `gaussianValue` of each pixel as seen in line 19 and returns a vector consisting of tuples with $x$ and $y$ coordinates for each key point. That is, we iterate through each heatmap to locate the area which holds the highest probability of matching the joint corresponding to that specific heatmap. Due to the risk of noise and

other irregularities in the output data, which at this point is unprocessed, and comes directly from the model, it is not necessarily sufficient to iterate through all pixels to find the one with the highest value as this could be placed at the outerskirts of the actual area of the given key point. It is for this reason that we opt for applying a Gaussian distribution inspired filter using the `getGaussian` function as shown in listing 6.8.

```
1  private static final float GAUSSIAN_WEIGHT = 0.2f;
2
3  private static float getGaussian(float[][][] heatmap, int bodypart, int row, int col){
4      float sum = 0;
5      // Iterating over the pixels adjacent to current.
6      for (int i = -1; i <= 1; ++i) {
7          for (int j = -1; j <= 1; ++j) {
8              int currentRow = row + i;
9              int currentCol = col + j;
10
11             float value = heatmap[currentRow][currentCol][bodypart];
12             float weight = i == 0 && j == 0 ? 1 : GAUSSIAN_WEIGHT;
13             sum += value * weight;
14         }
15     }
16     return sum;
17 }
```

**Listing 6.8:** Java code for computing Gaussian-value for pixels.

It should be noted that since we are not using a normal distribution, our values will not necessarily be between 0 and 1. We opt for not doing this, as it would introduce unnecessary computations; the point with this function is only to locate the most accurate $x$ and $y$ coordinates for the given key point. Indeed, since a heatmap is essentially a bivariate kernel density estimation—that is, each pixel in the heatmap has a certain area of influence, or kernel, which decreases the further we go from the key point [88, 89]. What happens in `getGaussian` is thus to compute the sum of the adjacent pixels to the pixel we are currently checking. This means we end up having a sum for each $3 \times 3$ submatrix throughout the heatmap. The one with the highest sum is marked as the key point. Ultimately, once this has been completed, we have an output similar to what is shown in fig. 6.6.

With the changes made on the Android side, we achieve inference times of approximately 70ms on the OnePlus 5T, which is more than a 50% reduction compared to the initial implementation, and even more on image conversions. In short, this translates to approximately 15 frames per second, which exceeds our requirement with a minimum of ten frames per second. To summarise; we meet our performance requirement using Apple's Core ML for iOS devices with ten frames per second for an iPhone 6 and 40 for an iPhone X, and by using the nightly build of TFLite with GPU acceleration for Android devices with 15 frames per second on a OnePlus 5T.

The sequence diagram in fig. 6.7 shows the entire process from the `onFramePreview`-function shown in listing 6.1 to returning the output vector consisting of tuples through the RNB to the React Native codebase such that subequent repetition detection and evaluation can be computed.


| | X | Y |
|---|---|---|
| 0 | 56.25 | 660.0 |
| 1 | 191.25 | 700.0 |
| 2 | 225.0 | 560.0 |
| 3 | 315.0 | 480.0 |
| 4 | 405.0 | 580.0 |
| 5 | 225.0 | 800.0 |
| 6 | 348.75 | 880.0 |
| 7 | 405.0 | 800.0 |
| 8 | 495.0 | 580.0 |
| 9 | 585.0 | 400.0 |
| 10 | 675.0 | 240.0 |
| 11 | 528.75 | 720.0 |
| 12 | 731.25 | 720.0 |
| 13 | 933.75 | 660.0 |

**Figure 6.6:** Labelled key points.

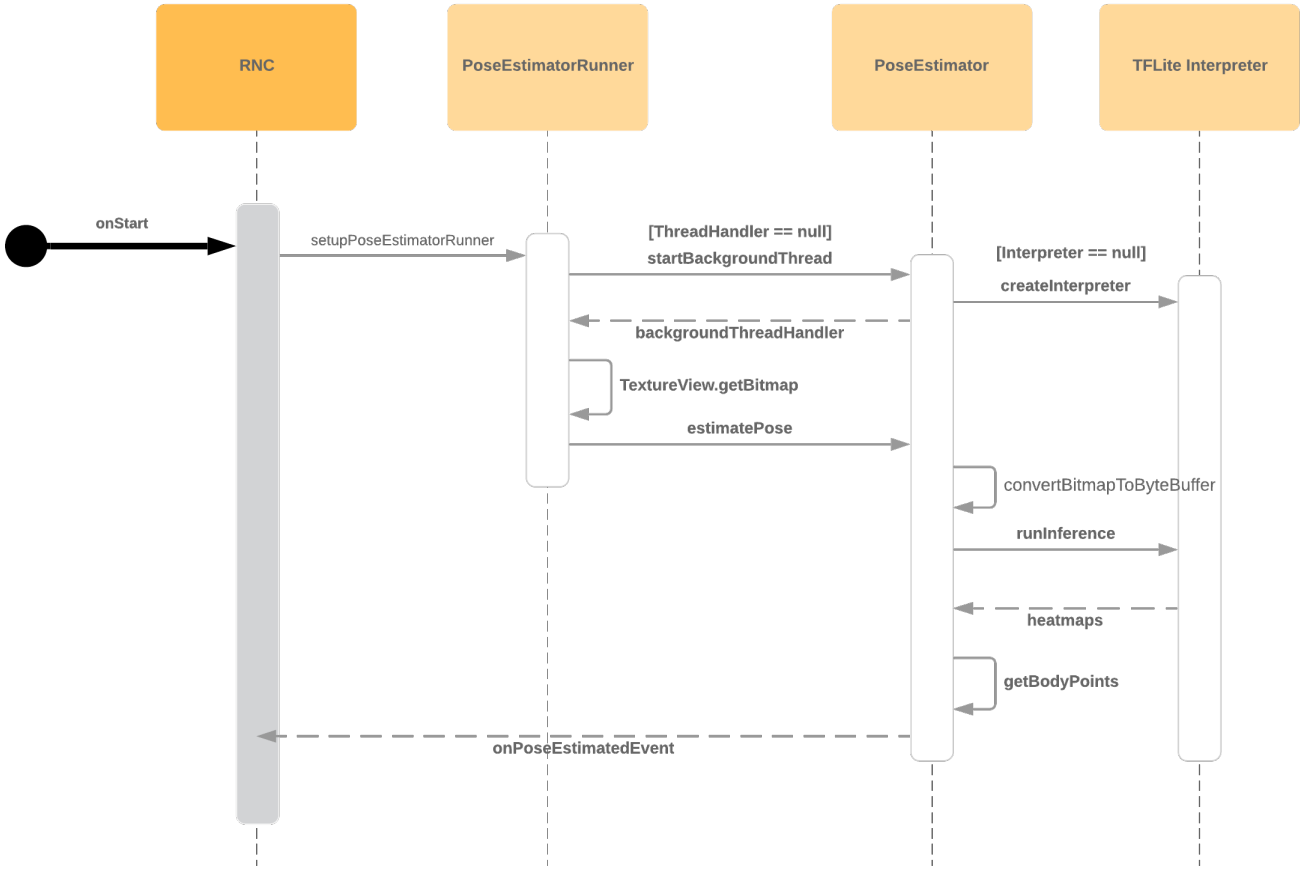



**Figure 6.7:** Sequence of operations from receiving a frame.

## 6.4 Repetition Detection and Evaluation Implementation

We have updated the system overview as seen in fig. 6.8 with the changes described throughout the previous section. It is in this regard important to accentuate that we now move from the native side of things to the React Native side. This also means that we, in this part, no longer rely on native-specific languages, but instead use TypeScript. The reason for using TypeScript instead JavaScript is to leverage the type system of TypeScript, which eases the process of concurrent development by enforcing consistency with input and output.

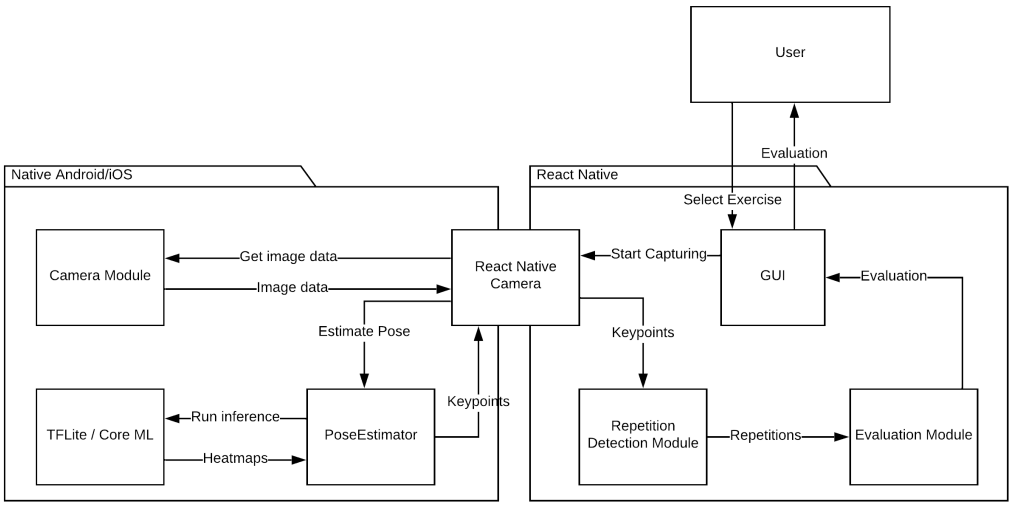



**Figure 6.8:** Updated system overview.

We now consider the two remaining modules of our design as outlined in chapter 5, namely (1) repetition detection, and (2) repetition evaluation. In accordance with requirement 8, *FormAssist* must provide repetition detection and evaluation in real-time as illustrated in fig. 5.1. This, inevitably, means the two modules are closely connected. We will, therefore, update our design accordingly and consider this a single, cohesive module moving forward. With this in mind, we propose the outline of the module seen in fig. 6.9.

The class diagram employs a hierarchical pattern consisting of five classes and includes an abstract `Exercise` class. The bottom of the hierarchy consists of the `Frame` and `Evaluation` classes. The `Frame` class consists of the input used for detecting repetitions and evaluating them. While frames are initially served to the model as a stream, they are aggregated into a specific repetition once one is detected. This means that when an exercise is complete, it is possible to determine which frames are part of which repetition. This is important because repetitions are evaluated individually. Depending on the exercise, a repetition will hold some number of `Evaluation` objects. The amount of objects per repetition depends on the exercise because the number of aspects that are evaluated depends on the exercise. For leg abductions, for instance, *FormAssist* will evaluate the angle between the user's legs during the positive and the negative phase of a repetition, as well as, whether both legs are

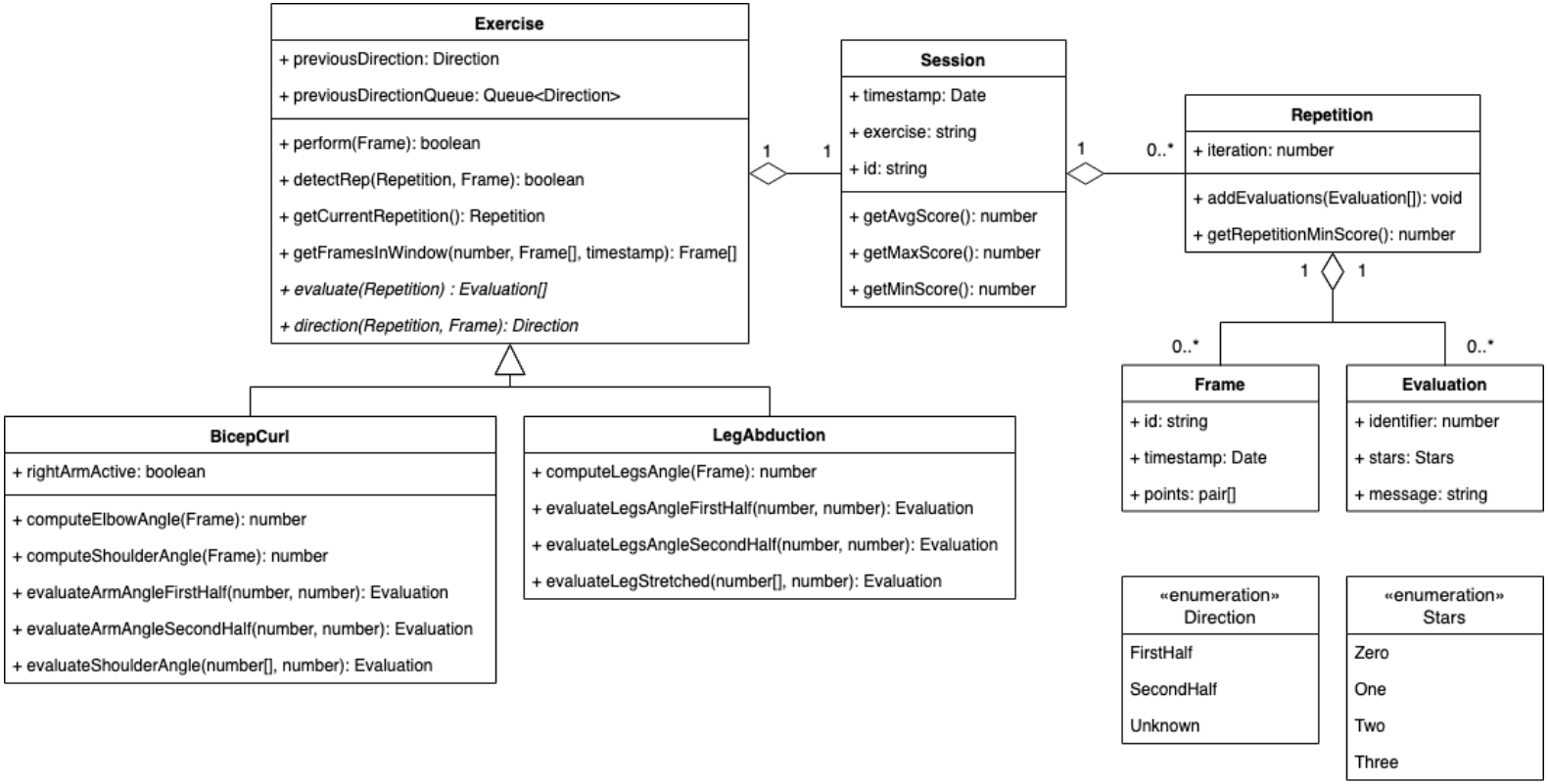



**Figure 6.9:** Proposed class diagram for repetition detection and evaluation.

stretched throughout the entire repetition. This means that four `Evaluation` objects are created for a single leg abduction repetition. We specifically evaluate the positive phase angle, the negative phase angle, and whether both the right and left leg remain extended throughout the repetition.

The primary properties of the `Evaluation` class are `stars` and `message`. They contain textual feedback detailing how well a user performed an exercise. The `stars` property is a score from zero to three. If a user performs some aspect of an exercise well during a repetition they will receive a score of three stars. If one or more mistakes are detected, they will instead receive a score of zero to two stars depending on how severe the mistake is. The severity is determined by a threshold which is set specifically for each aspect of an exercise. As an example, if a user keeps their right leg straight throughout a leg abduction repetition, they will receive three stars. If they at some point bend their leg, they will instead receive zero to two stars depending on the percentage of frames their leg was bent during the repetition. An `evaluation` object also consists of a message which explains which mistakes were detected for a given aspect of the exercise, and how the user can improve upon it. Similar to the score, the message will also depend on severity. As an example, if a user had their leg bent during most of a leg abduction repetition they will receive the following message: *"Your leg was not extended during most of the repetition. In this exercise your legs should be fully extended."*.

The `Repetition` class consists of a number of `Frame` and `Evaluation` objects. This class contains methods for determining the minimum scores achieved for a repetition in accordance with our design choice of aggregating using minimum scores. The `Repetition` class is aggregated by the `Session` class. A performance of an exercise is considered a session, and consists of multiple repetitions. Similar to the `Evaluation` class, the `Session` class includes a method for determining the average score of all repetitions belonging to the session. Since a `Repetition` holds an evaluation for each aspect of the exercise, and a `Session` holds multiple repetitions, the total number of evaluations quickly increases. As an example, four `Evaluation` objects are created for a single repetition of the leg abduction exercise. If a user performs 15 repetitions during a session, this will result in a total amount of 60 evaluations. Therefore, including functionality for viewing maximum, minimum, and average scores of a session is beneficial because it allows a user to gain insight into their performance, without having to examine the evaluations of all repetitions.

At the top of the hierarchy is the `Exercise` class, which contains functionality for detecting repetitions and evaluating them. As mentioned in section 5.2 and section 5.3, a challenge of implementing repetition detection and evaluation is determining how much functionality can be generalised to a generic exercise class, and what functionality should be exercise specific. Whilst we, initially, attempted to generalise most of the functionality to the `Exercise` class, we realised, during the implementation of the two child classes, `LegAbduction` and `BicepCurl`, that this was not feasible. Both repetition detection and evaluation is thus handled by the `Exercise` method `perform`. This method takes a frame, which consists of key points of a detected pose, as input and determines whether a repetition is completed. Once a repetition has been detected, it is evaluated and a `Evaluation` object is added to the active `Repetition` object. A new `Repetition` object is created and set as the current repetition. The input frame is then appended to the new `Repetition` object. If a repetition is not detected, no new objects are created and the frame is appended to the current repetition.

### 6.4.1 Repetition Detection

As described in section 5.2, we opted for a direction based approach to repetition detection in which an angle is used to check whether a limb is in a positive phase or a negative phase. The underlying idea is that a repetition must, inherently, contain both phases. As an example, during a leg abduction repetition, the user will start by extending their leg and then contracting it again. We can determine that a repetition has finished, once the user begins to extend their leg again. Checking the angle between the user's legs can be used to determine whether a positive or negative phase is currently being performed: If the angle is increasing compared to previous frames, the user is currently performing an extension. On the other hand, if the angle is decreasing, a contraction is being performed. However, generalising the required functionality from the leg abduction exercise to other exercises is tricky for the following reasons (1) using angles is not necessarily the optimal approach for detecting repetitions for all types of exercises, and (2) some exercises begin with a positive phase whilst others begin with a negative phase.

The first point is relevant for some compound exercises that require a wide range of motion such as burpees. For such exercises, checking key point positions may be more appropriate than checking angles between different body parts. In any case, implementing repetition detection such that it is possible to use different types of checks is preferred because one of the key attributes of *FormAssist* is extensibility. In short, it should be possible to implement many different types of exercises. The second point is a subtle, but significant variation in exercises that makes a generic implementation of repetition detection more cumbersome because it must be possible to determine which kind of movement is used during the positive and the negative phase of a repetition. Furthermore, for some types of exercises, such as burpees, it does not make sense to talk about extensions and contractions as a way to determine which phase is being performed since both phases may contain both extensions and contractions. For this reason, we decided to use the enumeration type `Direction` which can be seen fig. 6.9. We use `FirstHalf` and `SecondHalf` instead of `PositivePhase` and `NegativePhase` in order to make the difference easier to understand.

As a result of the issues discussed above, we implement the responsibility of determining a direction of an exercise—i.e. which phase a user is currently performing—to the specific implementation of each exercise. The `Exercise` class, therefore, contains an abstract method `direction` which all child classes must implement. This method compares a frame to previous frames using an exercise specific heuristic. The implementation of the `direction` method in the `LegAbduction` class can be seen in listing 6.9. The method retrieves the `Frame` objects that were captured during the last 500 milliseconds—which is an arbitrarily chosen value to avoid comparing only the current frame to the previous one—of the current repetition, and stores them in `controlFrames` in line 4. Using the `computeLegsAngle`, the method then computes the angle between both legs of the user and stores it in `legAngle` in line 6. In line 7, the current duration of the repetition, stored in `repetitionTime`, is computed based on the timestamp of the current frame and the first frame of the repetition. In order to determine the direction of the repetition in the current frame, `direction` will first check for null values in line 10. If `legsAngle`, or half of the angles computed from frames in `controlFrames`, are null, `Direction.Unknown` is returned. This case occurs if the key points cannot be determined from the frame. Then `repetitionTime` is checked to see if 500 milliseconds have passed. If this is not the

case, it is assumed that the repetition has just started and `Direction.FirstHalf` is returned. If more than 500 milliseconds have passed, `direction` will determine the current direction of the exercise using the methods `checkIfSecondHalf` and `checkIfFirstHalf`. These methods will compare `legsAngle` to angles computed from frames in `controlFrames` and determine whether `legsAngle` is the smallest or largest angle, and either `Direction.FirstHalf` or `Direction.SecondHalf` is returned accordingly. If a direction cannot be determined, `Direction.Unknown` is returned.

```
1  direction(repetition:Repetition, currentFrame:Frame) : Direction {
2      // Number of milliseconds used to get frames for comparison with current frame
3      const framesWindow:number = 500;
4      const controlFrames:Frame[] = this.getFramesInWindow(
5          framesWindow, this.getCurrentRepetition().frames, currentFrame.timestamp);
6      const legAngle = this.computeLegsAngle(currentFrame);
7      const repetitionTime:number = repetition.frames.length > 0 ?
8          currentFrame.timestamp.getTime() - repetition.frames[0].timestamp.getTime() : 0;
9
10     if (isNull(legAngle) || controlFrames.filter(
11         x => isNull(this.computeLegsAngle(x))).length > controlFrames.length/2){
12         // Handle too many nulls
13         return Direction.Unknown;
14     } else if (repetitionTime < framesWindow) {
15         // Handle too few frames in rep
16         return Direction.FirstHalf;
17     } else if (this.checkIfSecondHalf(controlFrames, legAngle)) {
18         return Direction.SecondHalf;
19     } else if (this.checkIfFirstHalf(controlFrames, legAngle)) {
20         return Direction.FirstHalf;
21     } else {
22         return Direction.Unknown;
23     }
24 }
```

**Listing 6.9:** The direction method implementation for the leg abduction exercise.

Once the direction has been determined by the `direction` method, it is returned to the `detectRep`as seen in listing 6.10. This method uses the class property `prevDirectionQueue` in order to maintain which directions have been returned from the `direction` method. The if statement in line 4 maintains a fixed length of the queue. The constant `QUEUE_LENGTH` is set to 7. This specific value is a tradeoff between confidence in a change of direction and latency, and should, therefore, ideally, be set based on the hardware of the device. As with `framesWindow`, we will return to this in chapter 8. In line 8, the return value of `direction` is enqueued in `prevDirectionQueue`, and the direction is subsequently determined using the `getDominantDirection` method. This method returns which type of direction is dominant in `prevDirectionQueue`—that is, if the queue consists of four `Direction.FirstHalf` and three `Direction.SecondHalf`, `Direction.FirstHalf` will be returned. In line 11, `detectRep` checks if the direction has changed from `Direction.SecondHalf` to `Direction.FirstHalf`. If this is the case, a new repetition has started and `repDetected` is set to true. Once this happens, *FormAssist* has detected a repetition. A consequence of this implementation is that detecting a repetition, inherently, requires a new repetition to begin—something which means

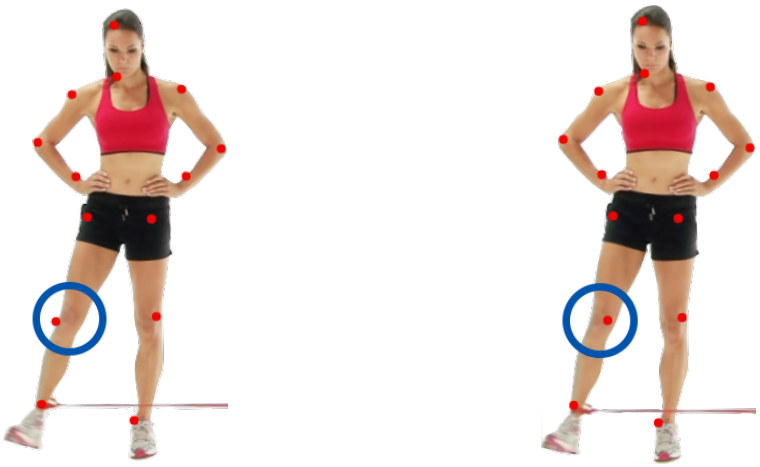
that we cannot detect the last repetition.

```
detectRep(repetition:Repetition, currentFrame:Frame) : boolean{
    let repDetected = false;

    if (this.prevDirectionQueue.count() >= QUEUE_LENGTH){
        this.prevDirectionQueue.dequeue();
    }

    this.prevDirectionQueue.enqueue(this.direction(repetition, currentFrame));
    const direction = this.getDominantDirection();

    if(direction === Direction.FirstHalf &&
        this.previousDirection === Direction.SecondHalf){
        this.fillQueue(direction);
        repDetected = true;
    } else if (direction === Direction.SecondHalf &&
        this.previousDirection === Direction.FirstHalf){
        this.fillQueue(direction);
    }

    if(direction !== Direction.Unknown){
        this.previousDirection = direction;
    }

    return repDetected;
}
```

**Listing 6.10:** The detectRep method of the exerciseDefinition class.

### 6.4.2 Key point Smoothing

As a preventative measure to combat possible irregularities in the output of the pose estimation model, we choose to proactively apply smoothing of the key point coordinates in order to ensure more consistent data. This is important because the detection of repetitions and subsequent evaluations rely upon the precision of the data. If there are too many imprecisions in the key point data, the quality of repetition detection and evaluation, inherently, decreases which, by natural extension, also means that the usefulness of *FormAssist* decreases. Imprecisions occur because the pose estimation model is not entirely accurate, and key points occasionally move irregularly. Consider, for instance, fig. 6.10a and fig. 6.10b which show two consecutive frames of a leg abduction repetition. Notice how the key point marking the left knee moves significantly compared to the actual position of the knee.



**(a)** frame one **(b)** frame two

**Figure 6.10:** Two frames of a leg abduction repetition.

In order to smooth the key point data, we employ the Savitzky-Golay filter, which smoothes the data by returning a weighted average of recent data points [90]. We have rewritten the original algorithm to handle streams instead of a complete data set. The algorithm for smoothing the key point data is given by the following equation:

$$y = \frac{\sum_{i=0}^{n} A_i \cdot Y_i}{\sum_{i=0}^{n} A_i} \tag{6.1}$$

Where $Y$ is the most recent data points, and $A$ is the set of their corresponding weights. Smoothing is handled by the class `Smoother`, which contains a method `smooth`. This method takes as input the coordinates of all key points in a frame and returns the updated coordinates after they have been smoothed. The result of computing the angle between both legs of a person performing three leg abduction exercises without smoothing the key point coordinates is seen in fig. 6.11. The x-axis shows the frames, and the y-axis shows the resulting angles in degrees. The result of computing the same angles but after the key points have been smoothed using the Savitzky-Golay filter is seen in fig. 6.12. Notice how the irregular spikes at frames 50, 98, and 133, respectively, have been reduced, and that the rate of change has been made more consistent.

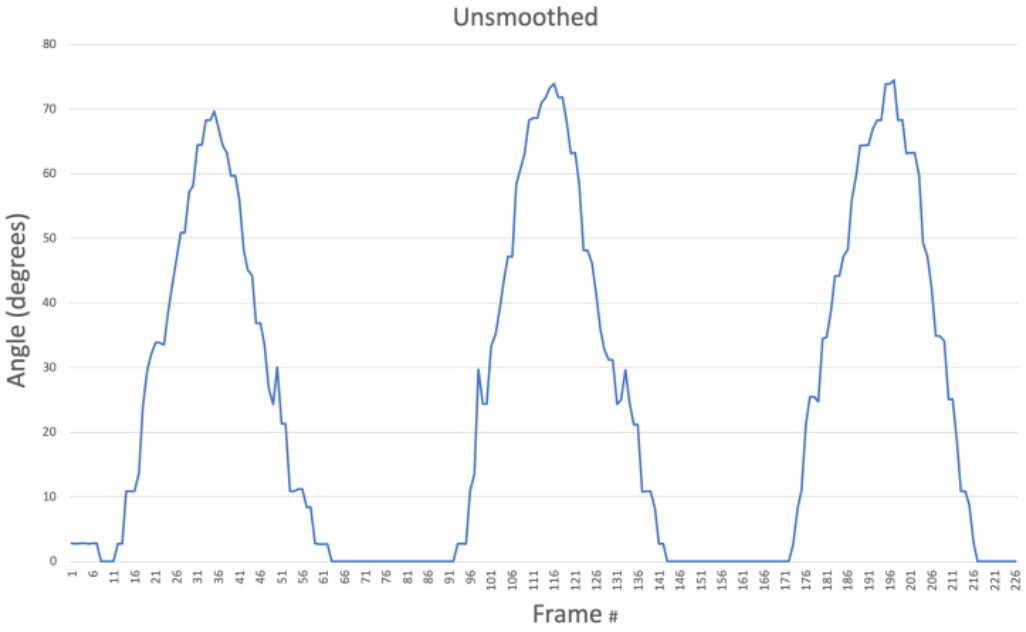



**Figure 6.11:** Unsmoothed angles during three leg abduction repetitions.

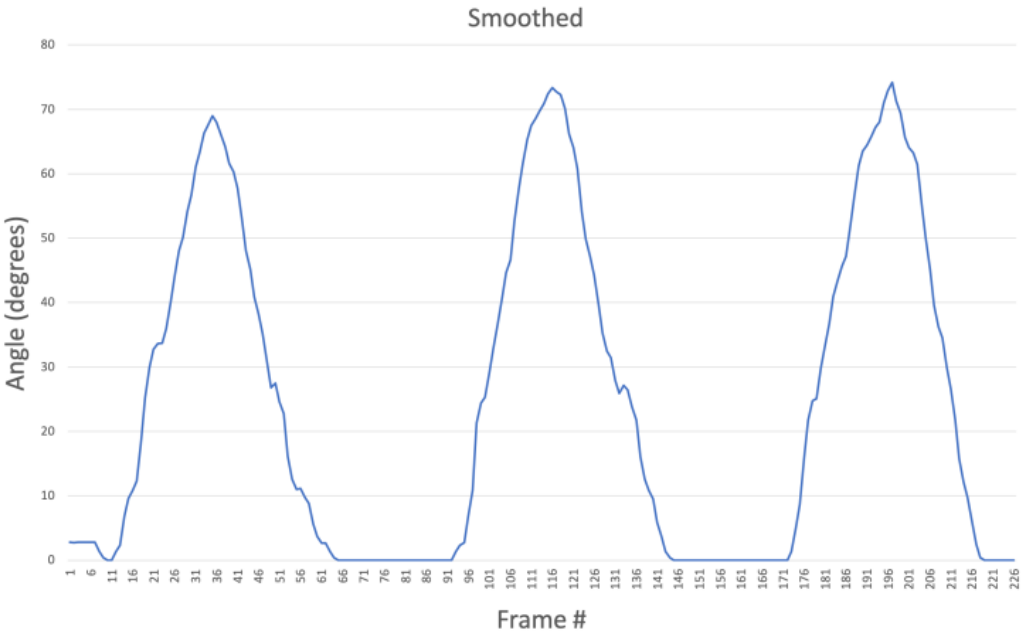



**Figure 6.12:** Smoothed angles during three leg abduction repetitions.

### 6.4.3 Repetition Evaluation

Once a repetition has been detected, we proceed to evaluate it. Depending on the exercise in question, and its associated heuristics, we evaluate $x$ different aspects of the given repetition. The evaluation of a repetition is handled by the `evaluate` method which is implemented on the specific `Exercise` class—i.e. `LegAbduction`. That is, the `evaluate` method is implemented based on the exercise specific heuristics. This method takes a `Repetition` object, containing all the frames identified as belonging to a single repetition, as input, and returns a list of `Evaluation` objects.

The implementation of `evaluate` for the `LegAbduction` class can be seen in listing 6.11. While this method is the only required method for repetition evaluation, we have isolated the responsibility of each aspect of an exercise into multiple methods. In lines 4-13, the maximum and minimum angles that the user reached during a repetition is computed and subsequently used for evaluation. When computing the minimum angle, only angles during the negative phase of the repetition are checked by filtering out all angles prior to the maximum angle of the repetition. In lines 15-22, the length of both legs during the entire repetition is computed in terms of their relative length compared to the torso of the user. These values are then evaluated in order to determine whether the user keeps their legs straight during the entire repetition. Using relative lengths instead of absolute lengths ensures that the values do not change if the user moves closer to, or further away, from the camera during an exercise.

```
1   evaluate(repetition:Repetition) : Evaluation[] {
2       ...
3       let evaluations:Evaluation[];
4       let legsAngles:number[] = repetition.frames.map(x => this.computeLegsAngle(x));
5       let reachedAngleFirstHalf:number = Math.max(legsAngles);
6       let reachedAngleSecondHalf:number = Math.min(legsAngles.filter(
7           (x, index) => index > legsAngles.indexOf(reachedAngleFirstHalf)
8       ));
9       evaluations.push(
10          this.evaluateLegsAngleFirstHalf(reachedAngleFirstHalf, legsAngleMaxId)
11      );
12      evaluations.push(
13          this.evaluateLegsAngleSecondHalf(reachedAngleSecondHalf, legsAngleMinId)
14      );
15
16      let rightLegtorsoRatios:number[] = repetition.frames.map(
17          x => this.torsoRatio(x, BodyParts.RightFoot, BodyParts.RightHip)
18      );
19      let leftLegtorsoRatios:number[] = repetition.frames.map(
20          x => this.torsoRatio(x, BodyParts.LeftFoot, BodyParts.LeftHip)
21      );
22      evaluations.push(this.evaluateLegStretched(rightLegtorsoRatios, rightLegId));
23      evaluations.push(this.evaluateLegStretched(leftLegtorsoRatios, leftLegId));
24
25      return evaluations;
26  }
```

**Listing 6.11:** The evaluate method implementation for the leg abduction exercise.

`evaluateLegsAngleSecondHalf` is responsible for creating objects relating to the evaluation of the angle that the user reaches during the negative phase of a repetition, and can be seen in listing 6.12. This method uses a threshold, `targetAngle`, to determine how well the user performed this aspect of the repetition. If the user reaches an angle equal to, or lower than, the threshold during the negative phase, they will receive three stars since this is considered an optimal performance. If a detected repetition scores two or three stars, a designated sound cue will be played for the user such that they know their most recent repetition was correct. This is how we opt for supporting real-time feedback as per requirement 8. If, however, the user performs subpar, they will instead receive zero or one star depending on how close to reaching the threshold they were. In this case, the real-time feedback will be a different sound cue indicating that the repetition was inadequate. Similarly, the message that the user receives as part of the post-performance evaluation is also determined by how close they were to the threshold. This message includes both what angle the user should reach and what angle they actually reached. This is handled by the `printAngles` method. Since `targetAngle` is set to 10, if the user then reaches 8 degrees, the method will return the following string: *"Target angle is 10 degrees. You reached 8 degrees."*.

```
evaluateLegsAngleSecondHalf(reachedAngle:number, id:number) : Evaluation {
    /* The angle that the user should be under
    during the second half of the repetition. */
    let targetAngle:number = 10;

    if (reachedAngle <= targetAngle) {
        return new Evaluation(id, Stars.THREE,
            "You contracted your leg far enough. Good job! " +
            this.printAngles(targetAngle, reachedAngle));
    }
    else if (reachedAngle <= targetAngle + 5)
        return new Evaluation(id, Stars.TWO,
            "You should contract your leg slightly more. " +
            this.printAngles(targetAngle, reachedAngle));
    }
    else if (reachedAngle <= targetAngle + 15) {
        return new Evaluation(id, Stars.ONE,
            "You should contract your leg moderately more. " +
            this.printAngles(targetAngle, reachedAngle));
    }
    else {
        return new Evaluation(id, Stars.ZERO,
            "You should contract your leg significantly more. " +
            this.printAngles(targetAngle, reachedAngle));
    }
}
```

**Listing 6.12:** Implementation of the method for evaluating the negative phase of a repetition.

A key difference between exercises is that for some of them, such as leg abductions, the user must record themselves from the front while for other exercises, such as bicep curls, the user must record themselves from the side. This is determined based on the heuristics that are used to evaluate an exercise. As an example, for the bicep curls exercise, *FormAssist* will evaluate the angle at the elbow

in order to determine the user's range of motion as well as how still the user is keeping their upper arm. Evaluating this is easiest when using the angle between the upper arm and the torso when the user is recorded from the side. This is because movement of the upper arm is easier to capture from this perspective. In order to determine which arm is being recorded when performing bicep curls, the `BicepCurl` class holds an extra property, `rightArmActive`, which must be set when an instance of the class is created. This is necessary because the correct key points must be used for both repetition detection and evaluation and it is not immediately obvious whether the key points of the right or left arm should be used based on the output of the pose detection model.

## 6.5 Database

When an exercise has been performed, we display the evaluation of the performance. In order to display previous evaluations we, naturally, must store the data. We faced two possibilities in terms of persistent storage for a mobile device; (1) storing our evaluations directly on the phone in its file system, or (2) using a server or some cloud service. However, as we have no need for such a service in any of the other modules of *FormAssist*, and its overhead on data transfer between a mobile device and an external server, we chose to use the local storage of the mobile device.

Having chosen our storage method, we are left with another, similar decision— on Android, there are four possible ways of handling persistent storage of application relevant data on the device, namely (1) internal storage, (2) external storage, (3) shared preferences, and (4) a database [91]. The method chosen comes down to a number of things; (a) how securely the data needs to be stored i.e. how sensitive the data is, (b) whether the user needs to be able to view it outside the application, and (c) whether an abstraction layer i.e. in the form of a Database Management System (DBMS), is needed.

By default, files saved to the internal storage are private to the application and are inaccessible by other applications and the user, unless they have root access. The user does not need to view the data saved outside of the application which makes this form of storage a good option. We can, immediately, dismiss external storage because it is not guaranteed to be accessible as it usually is in the form of a SD memory card or another plug-in like storage component. Additionally, the data can be modified by the user. Although this is not necessarily a problem, we would like to avoid the user inadvertently removing or modifying the saved evaluations. The third option, shared preferences, can also be dismissed as it is a primitive storage type and is used, primarily, to store mappings, or preferences, such as which ringtone should be used for a specific user. The last option we consider is; a database. Android provides full support for SQLite databases. Any database created is only accessible by the application, meaning other applications and the user cannot access it. Furthermore, it allows us to use a DBMS, which (1) allows us to attain type checking, (2) not having to deal with in which directories to store evaluations, which is the case with internal storage, (3) not handling the format of files. With this in mind, we proceeded to use a database as the functionality and customisation provided outweighs the other options available.

Having chosen our method of storage, we proceeded to research various DBMS' for React Native, as we are developing a cross-platform solution and, thus, need the DBMS to work across both Android and iOS. A member of the group had previous experience working with Realm [92] and, thus, suggested using it. Realm Database is a full-featured, object-oriented, cross-platform database that persists data locally on the device [93]. It is capable of handling very large data loads which, although not necessarily a feature we need in the context of a MVP, is worth to take into account with our extensibility attribute in mind.

### 6.5.1 Using Realm

Once Realm is installed and imported into the React Native project, setting up a database schema—i.e. the structure of the data we want to save—is done by defining an object as seen in listing 6.13. This is the database schema for storing a `Session`, which was discussed in section 6.4. A name, and optionally a primary key, is defined, as well as, the properties for the class. The property `repetitions` refers to the schema created for the `Repetition` class.

```
1  const sessionSchema = {
2      name: 'Session',
3      primaryKey: 'session_id',
4      properties: {
5          session_id: 'string',
6          timestamp: 'date',
7          repetitions: 'Repetition[]',
8          exercise: 'string',
9      }
10 }
```

**Listing 6.13:** Schema for the Session class.

Once the database schemas are defined, we can begin to open a connection to the database in order to insert and retrieve data. Querying the database is done by opening a connection, creating a new Realm object using the full list of required schemas, as see in lines 2-12 of listing 6.14. Realm checks whether a database matching this configuration exists, and connects to it. Realm, moreover, employs lazy evaluation, which means that only the properties which are actually needed are loaded. This makes querying efficient because objects can be handled as though they are completely loaded. Lines 14-19 show how the connection is used to query the database for a session object using its id.

```
...
this.realm = new Realm(
    {
        schema: [
            pairSchema,
            frameSchema,
            evaluationSchema,
            repetitionSchema,
            sessionSchema
        ]
    }
);
...
async getSession(sessionId:string):Promise<Session>{
    const dbSession:dbSession =
        this.realm.objects('Session')
        .filtered(`session_id = "${sessionId}"`) as unknown as dbSession;
    return this.fromDbSession(dbSession);
}
```

**Listing 6.14:** Opening and querying the database.

Once the connection has been established, we can communicate with the database and load the data we need into *FormAssist*'s GUI when requested.

## 6.6 Frontend

The aim is for *FormAssist* to be a part of physiotherapists toolbox, which they can give their patients to aid them in performing their exercises. This means that the users for *FormAssist* are diverse and the graphical user interface must therefore be as user-friendly as possible. We provide an interface where the user can view previously performed exercises or select a new exercise to perform. While the user performs the exercise, *FormAssist* evaluates the process. When a session is completed the user is presented with an overview of the evaluation. From here, the user can return to the initial view. The views are built as follows:

- **Overview**: The user can select and view the evaluation of previously performed exercises. A large button in the bottom of the view navigates to the `Select Exercise` view.
- **Select Exercise**: Here the user selects which exercise to perform. The exercises are grouped by their primary body part, such that finding and selecting an exercise is as easy as possible.
- **Perform Exercise**: This view starts with a countdown where the user has time to place the phone and get into position. When the countdown reaches zero, the exercise begins. The front-camera feed is displayed on the screen, to aid the user in correct placement of themselves and the mobile device. A button, that reads *Cancel* until the first repetition is detected, is placed at the bottom of the view. After this the text changes to *Stop Exercise*. If *Cancel* is clicked, the user is returned to the `Select Exercise` view, whereas *Stop Exercise* navigates

to `Evaluation`.

- **Evaluation**: This view shows the various scores for the session, using stars from zero to three. At the top of the view we display three aggregated scores for the session followed by each repetition below. The aggregated scores for the session are *Overall*, *Worst*, and *Best*, where the overall is an average of each repetition. Tapping a repetition expands it where a more detailed evaluation is available. Here a click on the play button shows the animation for the recorded points. Going back from this view returns the user to the `Overview` view.

Figure 6.13 shows all views of *FormAssist* and how to navigate between them.

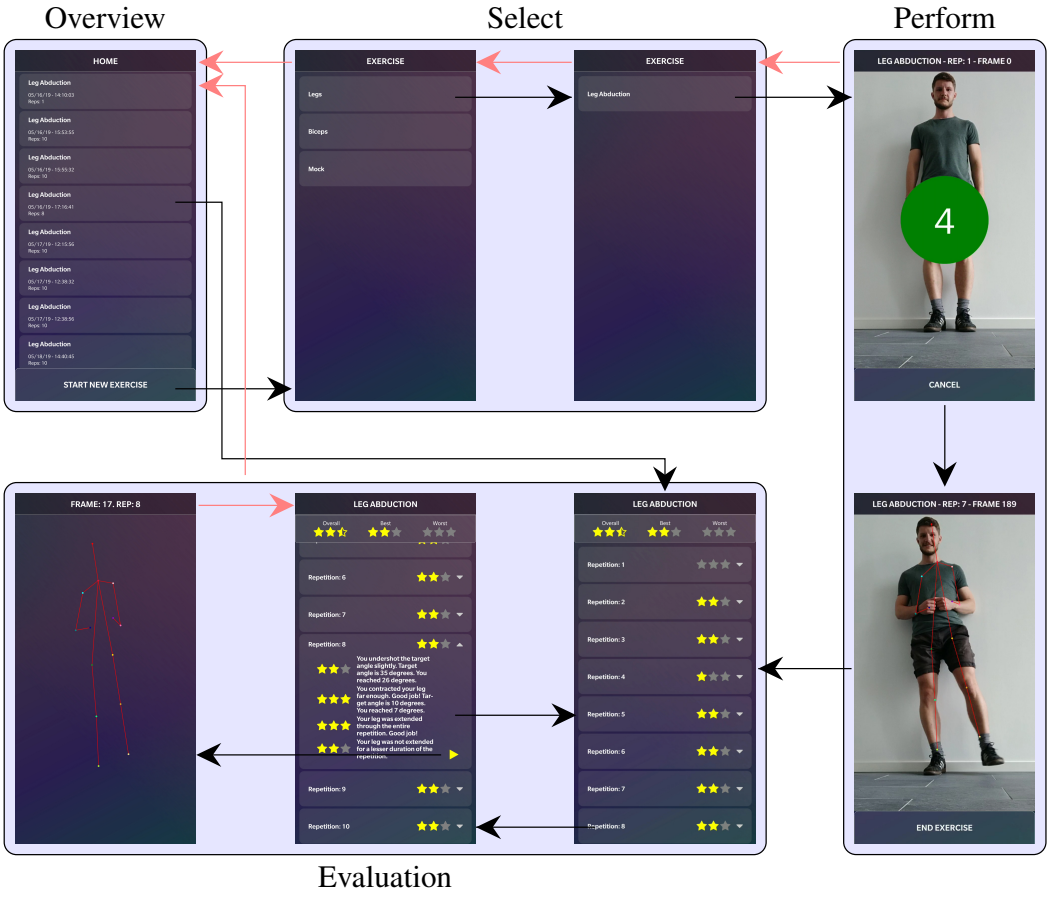



**Figure 6.13:** Navigation flow of *FormAssist*. Red arrows represents `Back` events

### 6.6.1 Application structure

React Native is based on the React API [94], which means it uses the same design. React uses composable components to create applications, and is primarily used for developing web applications and, thus, provides predefined components for all HTML tags [95]. React Native similarly includes components, but each component is an abstraction over a native component in Android and iOS.

These are provided without any styling, which means that most uses of a component require styling. The actual styling of the components is done using similar properties as those known from CSS [96].

A component can be created as a JavaScript class which inherits from the React Native Base Component. This base component gives us access to the React Native API inside the method. The `render` method is responsible for defining how a component should be displayed on the device. React uses an XML based syntax, called JSX, for this purpose [97]. JSX supports JavaScript code injection using the *{}* symbols. XML syntax is also used to define component properties. This binds values to a variable inside the component. An example of such a component can be seen in listing 6.15. We will prefix the names of components that we did not create ourselves with an underscore. Some of these are provided by React Native, and others are imported from JavaScript Packages. This is purely to aid the reader and is not a convention used in the codebase.

```
1  // The color variables are defined outside of
2  // the component
3  class Page extends React.Component {
4      render() {
5          return (
6              <_LinearGradient
7                  colors={[color3, color2, color1, color3]}
8                  start={{ x: 0.0, y: 0.0 }} end={{ x: 1.0, y: 1.0 }}
9                  style={styles.container}
10             >
11                 <_LinearGradient
12                     colors={[color4, color2]}
13                     start={{ x: 0.0, y: 0.5 }}
14                     end={{ x: 1.0, y: 0.0 }}
15                     style={styles.container}
16                 >
17                     {this.props.children}
18                 </_LinearGradient>
19             </_LinearGradient>
20         );
21     }
22 }
```

**Listing 6.15:** Code for the Page Component.

The `Page` component will be used as the outermost container for each of the views we described in fig. 6.13. This component displays the background for the application when rendered. We define the background with gradients using a component imported from a package called *react-native-linear-gradient*. The result can be seen in fig. 6.13. The gradients are controlled using properties as we discussed above. The `style` property in line 8 is used to add additional styling to the component such as setting the width and height. Please note that we omit styles object for all our components to keep code snippets more concise. Line 17 uses a special property which is created by React. The `this.props.children` returns a possible empty array of components that are nested inside the component itself. This pattern allows us to control where child components are to be displayed. The

first `_LinearGradient` component on line 6 has one child, which is another `_LinearGradient` component.

We create the following components in addition to `Page` to help build the views of *FormAssist*. These components allow us to adhere to the *don't repeat yourself* (DRY) principle [98].

- **H1, H2, H3, and Text**: By using these we avoid having to style every text element in *FormAssist* independently.
- **ListView**: React Native provides `_FlatList` to list a collection of similar items. We extend this with our own implementation thereof, called `ListView`, that requires a collection of data and a function to render each element. The additional properties required for `_FlatList` are then defined in `ListView`.
- **Element and ClickableElement**: To further simplify the `ListView`, we make `Element` and `ClickableElement` to avoid having to redefine the style of elements in a collection.

We can now start building the views using these components as well at those provided by React Native and other packages.

### 6.6.2 Integration with React Native Camera

We have discussed React Native Camera throughout section 6.3. We will now implemented the part of the frontend responsible for using RNC and controlling the pose estimation module. The `Perform` component can be seen in listing 6.16. We have removed some implementation details for the specific parts that will not be discussed—e.g. the definition of class properties, their initialisation, and some implementation of methods.

```
1  class Perform extends React.Component {
2      ...
3      poseEstimated = (obj: any) => this.poseEstimatedAsync(obj);
4      async poseEstimatedAsync(obj: { [x: string]: any; }) {
5          if (obj["bodyPoints"] === null) return;
6          let positions = scalePointsFromTo(96, 96, this.state.width, this.state.height, obj["bodyPoints"]);
7          positions = this.smoother.smooth(positions);
8
9          if (this.state.isPerforming) {
10             this.setState({ posePoints: positions, frame: this.state.frame + 1 });
11             let frame: Frame = new Frame(new Date(), positions, this.state.width, this.state.height);
12             const repEval: RepetitionCompletion = this.exerciseEvaluator.perform(frame);
13             if (repEval !== RepetitionCompletion.NOT_DONE) {
14                 this.setState({ rep: this.state.rep + 1 });
15                 if (repEval === RepetitionCompletion.DONE_CORRECTLY) {
16                     SoundHelper.playGoodJobSound();
17                 } else { // If done incorrectly
18                     SoundHelper.playBadJobSound();
19                 }
20             }
21         }
22     }
23     componentDidMount() {
24         this.timeoutHandle = setTimeout(() => {
25             this.setState({
26                 isPerforming: true
27             })
28         }, secondsToCountdown * millisecondsPerSecond);
29     }
30     render() {
31         let view = this.state.isPerforming
32             ? renderBody(this.state.posePoints, this.state.width, this.state.height)
33             : this.state.countdown;
34
35         return (<Page>
36             <Header>{this.exercise} - Rep: {this.state.rep} - Frame {this.state.frame}</Header>
37             <Body>
38                 <RNCamera
39                     ...
40                     onPoseEstimated={this.poseEstimated}
41                 />
42                 {view}
43             </Body>
44             {this.stopWorkoutButton()}
45         </Page>);
46     }
47 }
```

**Listing 6.16:** Code for the Perform Component.

The `RNCamera` component in line 38 is provided a modified version of the component provided by RNC. We have added the `onPoseEstimated` property on line 40, which expects a function,

that will be called every time the user's pose is estimated. There are additional properties that are defined by RNC; most of these properties are used to control various aspects of the smartphone camera— most of these have been left with default values and have, consequently, not been included in the snippet. The `RNCamera` component will display the output from the front facing camera when rendered. A repetition counter is shown above the camera to indicate the progress of an exercise. The `componentDidMount` method in line 23 is called by React when a component is mounted. That is, when a component is about to be displayed on the screen. The `componentDidMount` method is part of a collection of lifecycle methods controlled by the React base component [99]. These methods enable us to implement functionality that is called when a React event is triggered such as a component mounting. For our purpose, the `componentDidMount` will be used to display a countdown in the center of the screen. We call the built-in `setState` instead of displaying the countdown directly. Modifying the *state* of a React component will cause it to re-render, which calls the `render` method again. The expression on line 31 is responsible for setting up the countdown. The countdown is displayed at the start of an exercise and is removed when `setState` inside the timeout in `componentDidMount` is called. The `setState` will update the `isPerforming` property to true, which causes the expression on line 31 to return the result of calling the `renderBody` method instead of the countdown. The `renderBody` method displays a stickfigure based on the key points provided from the `onPoseEstimated` event.

`poseEstimatedAsync` method in line 4 is responsible for handling the data provided by the pose estimation module. The key points are provided as an object with *bodypoints* as the key. We must scale the key points because the output of the model is a downscaled to a $96 \times 96$ matrix. To render the stickfigure in proper scale, we must scale the key points to the size of the screen. We then apply the smoothing filter as discussed in section 6.4.2. In line 10 in listing 6.16, we update the component state with new key points and increment the frame counter by 1. This counter is primarily used for debugging. A new instance of the `Frame` class is created in line 11. The definition of the `Frame` class can be seen in listing 6.17.

```
1  class Frame {
2      id: string;
3      timestamp: Date;
4      points: pairArray;
5      width: number;
6      height: number;
7
8      constructor(
9          timestamp: Date,
10         points: pair[],
11         width: number,
12         height: number,
13         id: string
14     ) {
15         this.id = id;
16         this.timestamp = timestamp;
17         this.points = points;
18         this.width = width;
19         this.height = height;
```

```
20        }
21    }
```

**Listing 6.17:** The Frame class definition

We use this class to associate a set of key points with a timestamp. We also store the dimensions of the screen that was used to scale the key points. The frame is used as input for repetition detection and evaluation. `repEval` contains the evaluation of a repetition as seen in line 12 in listing 6.16 is an enum that indicates the status of a repetition. A repetition will consist of multiple frames and we only need to provide feedback once a repetition is complete. In addition to playing a sound cue indicating performance, we also increment the repetition counter.

## 6.7 Summary

Having completed the implementation of the various modules of *FormAssist*, we now present the final system overview of the MVP in fig. 6.14.

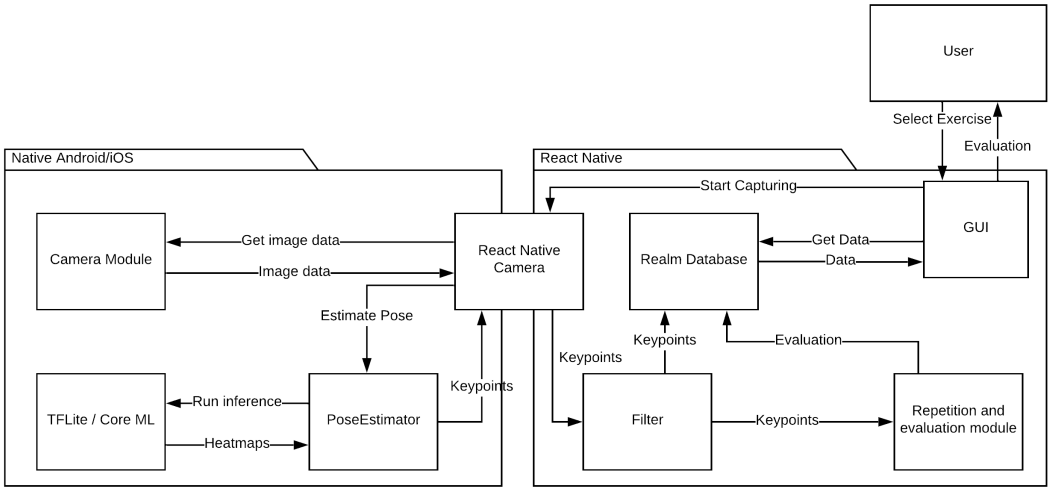



**Figure 6.14:** Finalised MVP system overview.

The MVP implementation of *FormAssist* consists of a native part, and a cross-platform, React Native part. Since we support both Android and iOS, we, effectively, have two implementations of the native side. The Android version relies on the beta version of TFLite with GPU support for inference, whereas the iOS version relies on Apple's Core ML. In both cases, we employ Edvard Hua's pre-trained model for pose estimation. The implementations on the native sides are, beyond the different machine learning libraries, identical.

The MVP of *FormAssist* thus supports the following functionality:

1. The user can select one of two exercises—either leg abduction or bicep curls—using the GUI.

2. *FormAssist* uses React Native Camera to send images from the front facing camera to the `PoseEstimator` as a user performs an exercise.
3. The `PoseEstimator`-class accepts image data as input and runs inference using either TFLite beta for Android, or Core ML for iOS, and processes the returned heatmaps before returning estimated key points.
4. *FormAssist* can detect repetitions and evaluate certain criteria of each detected repetition.
5. We can score the performance of an exercise with zero to three stars and provide textual feedback.
6. Evaluation of previous exercises can be retrieved from a local Realm database.

We now proceed to testing the MVP implementation.

# Chapter 7

# Testing

In accordance with our testing plan outlined in section 5.4, and following our implementation of a MVP-version of *FormAssist*, we now proceed to present the steps we have taken to test and otherwise verify that the current implementation adheres to our requirements established throughout chapter 4. In the abovementioned testing plan, we argued how the components related to the pose estimation module are deterministic, and how, as a direct consequence, we only require confirmation that the expected output is produced given a specific input. Since these steps were taken as part of the actual implementation—indeed, had any failed, we would have been unable to proceed with our development—the following sections will focus exclusively on the two remaining categories identified in the testing plan; namely repetition detection and evaluation as well as usability.

## 7.1 Repetition Detection and Evaluation

In testing *FormAssist*'s ability to detect repetitions and otherwise evaluate them correctly, we employ two types of tests, namely (a) unit tests, and (b) integration tests. The former will focus on individual functions, whereas the latter will focus on the module as a whole. Where unit tests will ensure our logic is correct, integration tests will ensure the module as a whole works as intended.

### 7.1.1 Unit Tests

The abstract super class, `Exercise`, shown in fig. 6.9, which each specific exercise inherits from, provides a set of helper functions used for evaluating proper form. These functions are generic, and mostly relates to the computation of angles and distances between key points. Since they are used throughout the evaluation functions, it is essential that they work correctly. A few of the unit tests for the function `computeAngleBetween` can be seen in listing 7.1. The `toBeCloseTo`-function seen in lines 6 and 15 is provided by the testing framework Jest [100], which we have employed for all of our unit tests. In this case specifically, `toBeCloseTo` is used as we need to compare floating point values.

```
...

test('angle 1 between (0,0), (15,0), and (16,5)', () =>
  {
    expect(body.computeAngleBetween([0,0], [15,0], [16,5]))
      .toBeCloseTo(101.31);
  }
);

...

test('angle 2 between (15,0), (0,0), and (16,5)', () =>
  {
    expect(body.computeAngleBetween([15,0], [0,0], [16,5]))
      .toBeCloseTo(17.35);
  }
);

...
```

**Listing 7.1:** Two unit tests for computing an angle between three points.

We develop a complete set of unit tests for every helper function offered by `Exercise` to ensure each function both works and fails as intended. It is, however, important to accentuate that this functionality of *FormAssist*, once in production, will only ever be called in the first place when the pose estimation module has generated an output.

### 7.1.2 Integration Tests

The very first step of conducting integration tests for the repetition detection and evaluation module is to find videos showing the two exercises—biceps-curl and leg abduction—supported by the MVP-version of *FormAssist*. We opt for this approach because we cannot reliably test any functionality if we were to continuously record ourselves performing a given exercise actually using the application. Thus, in order to consistently reproduce results, and run multiple tests using the same data, we develop a testing environment using various command line tools, e.g. youtube-dl [101] and ffmpeg [102], as well as a python script which loads our pose estimation model. This environment enables us to, essentially, mock the functionality of the smartphone camera and feed the pose estimation model a video of our choosing. The goal of this is to check whether it outputs the expected results—that is, the correct number of repetitions as well as the correct evaluations per repetition. We have, additionally, asked Mr. Jensen, one of the collaborating physiotherapists, to evaluate a video for each exercise. This evaluation is similar to the one *FormAssist* provides—i.e. stars from zero to three for each aspect of a repetition. We can use this to compare our evaluation results against an expert evaluation.

The process for each video is thus as follows; (a) load the video into our testing environment, (b) process the video as *FormAssist* would do natively on a smartphone, (c) run inference using the model for pose estimation on each supplied frame, and (d) smooth the output. At this point, we have a `.json` file containing the output of the supplied video. We then use this to detect repetitions before

| Repetition | Positive Phase | Negative Phase | Left Leg Straight | Right Leg Straight |
|---|---|---|---|---|
| 1 | 3 / 3 | 3 / 3 | 3 / 3 | 3 / 3 |
| 2 | 3 / 3 | 3 / 3 | 3 / 3 | 3 / 3 |
| 3 | 3 / 3 | 3 / 3 | 3 / 1 | 3 / 2 |
| 4 | 3 / 1 | 3 / 3 | 3 / 3 | 1 / 0 |
| 5 | 3 / 3 | 3 / 3 | 3 / 3 | 1 / 3 |
| 6 | 3 / 3 | 3 / 3 | 3 / 3 | 2 / 2 |
| 7 | 2 / 3 | 1 / 1 | 3 / 3 | 2 / 3 |
| 8 | 2 / 3 | 1 / 1 | 3 / 3 | 3 / 3 |

**Table 7.1:** The expected/actual evaluation scores for a leg abduction exercise with 30 frames per second. All underlined cells are the ones in which there is a discrepancy between the expected and actual scores.

sending it to our `Evaluation` component. This component returns an evaluation object on which we assert the correctness of specific properties. As seen in listing 7.2, the properties include the amount of repetitions, seen in line 1, alongside the scores, in terms of stars, for each type of evaluation we provide for each repetition, as seen in lines 2-6. For this specific example, we expect to detect eight repetitions, where the eighth has problems with both the positive phase and the negative phase, as indicated in line 6 by the scores 2 and 1. Please note that due to the MVP implementation of repetition detection, we have accounted for our inability to detect the last repetition by using a video containing nine repetitions.

```
1 this.assert(session.repetitions.length, 8);
2 this.evaluateRep(session.repetitions[0], [3, 3, 3, 3]);
3 ...
4 this.evaluateRep(session.repetitions[4], [3, 3, 3, 1]);
5 ...
6 this.evaluateRep(session.repetitions[7], [2, 1, 3, 3]);
```

**Listing 7.2:** Asserting the repetition count and evaluation scores for a leg abduction session.

The values used in this test are the ones defined by Mr. Jensen. The expected and actual values can be seen in table 7.1. The first digit of each cell defines the expected value, whereas the second digit defines the actual value. Those which differ have been underlined accordingly.

The fourth repetition is deliberately made with a bent right leg, which is what causes the evaluation score to drop to zero. The score for the positive phase is expected to be three, but as seen in fig. 7.1, the angle reaches more than 50 degrees, which is why the `Evaluation` module returns an evaluation of one. The positive phase in repetition seven and eight is evaluated to two by Mr. Jensen, since he assessed the angles to be too small. However, as seen in the graph, the seventh angle is larger than the first three. It could thus be argued that the cut-off should be around 35 degrees. This would make the eighth extension achieve an evaluation score of two in accordance with by Mr. Jensen's rating.

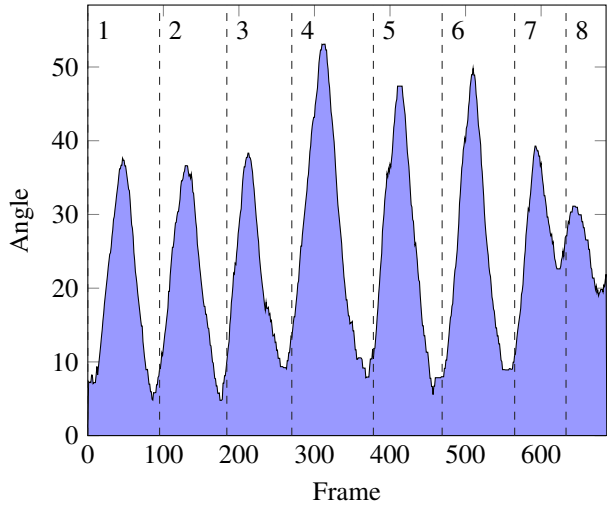



**Figure 7.1:** The computed angles during eight repetitions of leg abduction.

### 7.1.3 Testing Frames Per Second

In chapter 4, we estimated how *FormAssist* requires at least ten frames per second in order to adequately provide feedback. The testing conducted throughout the previous section used 30 frames per second—a frame rate enabled by the testing environment we designed—in order to give the best possible setup for approximating Mr. Jensen's evaluations. The absolute difference between Mr. Jensen's and *FormAssist*s evaluations for leg abduction at 30 frames per second sums to 11. We now run the same test with 5, 7.5, 10, and 15 frames per second in order to gauge how performance is impacted by the frame rate. Table 7.2 shows the results from running these tests.

| Frames per second | 5 | 7.5 | 10 | 15 | 30 |
|---|---|---|---|---|---|
| Repetitions detected | 7 | 7 | 8 | 8 | 8 |
| Summed absolute difference | N/A | N/A | 14 | 12 | 11 |

**Table 7.2:** Evaluation with different frames per second.

This is specific to both the leg abduction exercise and video used for testing, and whilst some exercises could have higher requirements, we are, nonetheless, able to verify that this exercise was adequately evaluated with a minimum requirement of ten frames per second. Anything lower than ten frames per second results in faulty repetition detection. This supports our initial decision of choosing ten frames as the minimum rate.

### 7.1.4 Usability Testing

As stated in chapter 4, we consider usability an important quality for *FormAssist*. Whilst we rely on physiotherapists to instruct users on how to both install and use *FormAssist*, it should still be user-friendly, such that users can easily navigate the application and access the provided feedback

following an exercise on their own. Due to the inherent time constraints of the project, and in the context of developing an MVP, we opt for not performing comprehensive usability testing. However, in order to attain a preliminary understanding of how usable *FormAssist* is, we will conduct some exploratory tests, which should indicate whether the MVP design is on the right track, and otherwise help improve the usability for future development. Indeed, usability, in the context of a release candidate of *FormAssist*, should inarguably be a top priority.

We will conduct a test based on the use-case described in chapter 4. After a brief introduction to the application, we will ask testers to complete the same use-case. For the purpose of conducting the test, we will ask testers to perform leg abductions as the exercise they have been prescribed. This means the testers should complete the following tasks:

1. Open *FormAssist* and navigate to the leg abduction exercise.
2. Start the exercise and position the smartphone as instructed.
3. Perform ten repetitions of the leg abduction exercise.
4. Review the provided feedback, and determine whether any issues were identified with their performance.

In order to evaluate the usability of *FormAssist*, we will observe if the testers are able to fulfil the tasks mentioned above as well as whether any complications arise. We will do this using the IDA method, which is designed to allow usability evaluations to be conducted and analysed in a single day [103]. In other words; we will test the flow of *FormAssist* and whether each step, as explained in fig. 6.13, naturally leads into the next one. For the last task, we will ask the testers to explain the feedback they have received in order to determine if the feedback is easily understood.

We conducted the test with two volunteers. We designed the test such that one of the test subjects received detailed instructions on how to use *FormAssist* and the other did not. The reason for this is that we designed *FormAssist* to be used as an expert application, i.e. users need instructions before using the application. However, we wanted to test whether these instructions are even necessary. Of course, it should be accentuated that two volunteers is not even remotely representative of the target group, and that both are experienced users of smartphones. Before starting the tests, both subjects were instructed in how to perform the leg abduction exercise. The IDA-method classifies the problems observed into three separate categories based on severity: critical, serious, and cosmetic. We classified our observations according to these categories, with our requirements in mind such that we can determine in which order we should fix the issues.

- Critical
    1. The feedback sound is delayed; especially if the person pauses at the end of a repetition.
- Serious
    1. It is not currently possible to detect pauses between repetitions.
    2. There was feedback for both right and left leg, but it was unclear which was which.
    3. Feedback regarding whether a leg was fully extended or not was unclear and confusing.
    4. A 5.5 inch screen is too small to determine whether you are at the right position.
- Cosmetic
    1. It is not immediately obvious how to navigate to a previous view in the application.

The primary issue we identified following the test, concerns when the sound used for real-time feedback is played. During some repetitions, with one of the test subjects, it played late relative to when their repetition had ended. This is due to the fact that repetitions are detected once a new one begins, and that feedback, by current design, is not provided until the user begins their next repetition. As a consequence, if one pauses following a repetition, a sound is, effectively, played right as one starts the next repetition. This was an understandable source of confusion for the test subjects because it was difficult to determine which repetition the sounds were related to. Another issue is the fact that pauses between repetitions cannot be detected. This means that if the user tries to adjust their position, *FormAssist* will still try to determine if a repetition has been performed. This is problematic both because it will lead to incorrect evaluations, but also because it will cause the real-time feedback to trigger a sound while the user is not actually performing a repetition. This means the sounds becomes disruptive, rather than helpful. We observed that a test subject felt the need to adjust their position because they believed that they were not fully captured by the camera when performing the exercise—even though this was not actually the case. This is a result of a 5.5 inch screen being too small to accurately determine whether the user has positioned themselves correctly.

We also discovered issues related to the feedback provided when an exercise has been completed. For the leg abduction exercise, feedback is provided describing whether both legs were fully extended during the entire repetition. This feedback is provided as two separate evaluations—one for each leg. It is not clear, however, which evaluations relate to which leg and the testers were, therefore, not able to determine how they can improve their performance. Furthermore, this issue led to confusion regarding what the feedback concerning the first place. One user arrived at the conclusion that the evaluations regarding keeping both legs extended were related to the same leg. This led to confusion because one evaluation stated that the relevant leg was fully extended throughout the entire repetition, while the other evaluation stated that the leg was not extended for a small duration. The user, therefore, concluded that they should keep their leg in the position reached at the end of the positive phase.

In terms of navigating *FormAssist*, the only problem we observed was that it features no button to return to a previous view. Smartphones running Android, usually, feature such a button by default,

though this is not the case for smartphones running iOS, where such a button is typically part of the application itself. Since the test was conducted using a smartphone running Android, this led to a small amount of confusion for one of the subjects, who usually uses a smartphone running iOS, because it was not immediately obvious to them how to navigate back to a previous view.

While we identified many important required changes, we also confirmed how many aspects of the user interface work as intended. Specifically, the test subjects had no issues understanding, nor browsing, the feedback provided after the session was completed. They were both quick to identify problems of their performances due to the star ratings supplied for each aspect of the exercise. They, furthermore, quickly understood the intention of the real-time feedback, and both attempted to repeat the movements they had used when the sound indicating a good repetition had been played. Whilst this, arguably, is a very confined, preliminary test, we, nonetheless, believe it suggests that the usability of *FormAssist* is on the right track. This is not to say that usability, nor the testing procedures, cannot be improved—we believe both can still be improved—but rather a preliminary confirmation that, in the context of a MVP implementation of *FormAssist*, we have developed a reasonable solution in terms of usability. This can provide insight into what a release candidate of *FormAssist* can achieve. Thus, in future work, we will make usability a high priority.

# Chapter 8

# Discussion

In chapter 4, we determined which software qualities are important for the development of *FormAssist*. At this point in the project we can, therefore, evaluate the overall quality of the current state of *FormAssist*, based on how well it adheres to the software qualities we selected. We, furthermore, established a number of requirements which we deemed necessary for a satisfactory MVP of *FormAssist*. We will, in the same vein as with the software qualities, evaluate whether these requirements have been successfully fulfilled. While no requirements established in section 3.2 are missing from the current state of *FormAssist*, they have been implemented with varying degrees of satisfaction.

*FormAssist*'s ability to estimate poses, as outlined in requirement 5, alongside its ability to detect repetitions as outlined in requirement 6 are, arguably, two of the most important requirements because many of the other requirements depend entirely on these being fulfilled. We are, therefore, satisfied with *FormAssist*'s ability to estimate poses, though whilst repetition detection is similarly supported, it is not currently in a state, which we deem completely satisfactory. The current repetition detection implementation works by determining which part of the exercise is currently being performed—i.e. the positive or the negative phase. By doing this, repetitions are detected when the user stops performing a negative phase, and switches to a positive phase. We opted for this implementation such that we maintain the ability to detect incomplete repetitions, should a user not reach a necessary requirement—e.g. minimum angle or minimum distance—for the given exercise. Indeed, in order to assist those who do not perform their exercises correctly, *FormAssist* inarguably needs to be able to detect incomplete and otherwise incorrect repetitions. The downside of the current implementation, however, is that *FormAssist* is unable to determine when an exercise session is completed because the last repetition is not detected—indeed, no positive phase follows the completion of the last repetition.

One approach we have discussed for potentially fixing this issue is to change the timing of when a repetition is detected. Instead of relying on a new repetition to begin for the previous one to be detected, it may be viable to, merely, check for consecutive frames featuring little to no movement. We argue that this may be sufficient to detect a repetition since a short period of time in which a user does

not move significantly is likely to follow a repetition—something which is further supported by how our test subjects, in our usability test, see section 7.1.4, behaved. It should be noted that this is only speculation and that substantial testing, consequently, should be conducted in order to verify whether this approach is viable. This issue should, consequently, be addressed accordingly in future development seeing how, if a repetition cannot be detected, it cannot be evaluated. This issue, furthermore, also negatively impacts requirement 8 concerning real-time feedback. As we also discovered in our usability test, the timing of the real-time feedback was a source of confusion for our test subjects because the sound signifying how well a repetition is performed, is played as the user has started performing the next one. Detecting repetitions earlier would, moreover, allow feedback evaluations to be computed earlier, thus allowing real-time feedback to be provided earlier. In addition to this, the current implementation of audio feedback of *FormAssist* is not what is defined in section 5.2. This is an architectural choice made during implementation, thus moving the responsibility of providing feedback from the specific exercise implementation to the GUI.

Computing repetition evaluations is based on the heuristics defined for each exercise as stated by requirement 2. The heuristics used by the current version *FormAssist* are defined by us, based on information found on the Internet regarding how to perform leg abductions and bicep curls correctly. This, essentially, means that the heuristics were not verified by a domain expert before they were implemented. A better approach would have been to consult the physiotherapists we interviewed at the start of the project in order to have them help us define heuristics before implementing them. However, as explained in chapter 7, we contacted Mr. Jensen and asked him to evaluate two videos of two group members performing leg abduction and bicep curl repetitions of various quality. Mr. Jensen verified that the heuristics we employ for these exercises were sufficient. The fact that we did not contact them earlier about defining the heuristics have, therefore, not had any negative impact on the usefulness of the MVP, though we still should have.

We should, however, in the context of further development make sure to involve physiotherapists in the process of defining heuristics for new exercises. In terms of developing a release candidate, this is essential because *FormAssist* is designed with physiotherapy in mind, and exercise definitions should, therefore, adhere to the way physiotherapists would define them. Having physiotherapists help us would not only ensure that the heuristics we use are correct, but also save us time in development, since we will not have to gather the information ourselves.

The last requirement which is not completely satisfactory is requirement 10 concerning evaluation. In section 7.1.4, we identified two issues regarding the feedback provided after a session: It was not immediately obvious that feedback about keeping legs extended during a leg abduction repetition is supplied for both legs. As a consequence of this, it was not obvious which aspect of the exercise the feedback was related to. In order to address these issues, the messages supplied as part of post-feedback should be reviewed and updated with input from physiotherapists. If an aspect is evaluated for both legs, or arms, it should be clearly indicated which limb an evaluation is related to. It is, in this regard, important to accentuate that the functionality of providing feedback is working as intended. It is, merely, the messages that are supplied with the evaluations that are problematic.

On the subject of requirements, we will, additionally, review our system quality attributes and investigate if, and to what extent, we have fulfilled them.

- **Extensibility**: We believe we have lived up to the attribute. *FormAssist* has been built, such that it allows us to extend its functionality seamlessly. The repetition detection and evaluation module allows for easy and generic setup of additional exercises, as was the intention based on requirement 2. Furthermore, the model we use for pose estimation can easily be substituted for another, should we wish to do so. Any limitations *FormAssist* may suffer from are, essentially, non-factors in that we can extend or swap out those limitations on-demand.
- **Portability**: We have developed a cross-platform application which runs on both Android and iOS, the two major mobile operating systems [33]. Additionally, *FormAssist* does not rely on any external services and runs completely locally. This allows the users of *FormAssist* to always have access to it regardless of where they are, and whether they have access to the Internet.
- **Usability**: Through the exploratory usability test of *FormAssist* explained in section 7.1.4 we discovered multiple issues which hampers this system quality. Since multiple of the issues were either critical or severe, we do not consider the usability of *FormAssist* to be satisfactory. Further development should, therefore, focus on remedying these issues as well as conduct further usability tests. Indeed, it should be noted that in order to ensure a high level of usability, a much more extensive test than what we have conducted in the context of an MVP is required. The test we conducted only included two test subjects of similar demographic. Since the intended target group of *FormAssist* effectively is all patients of physiotherapists, a proper usability test should, therefore, include more test subjects of varying ages and with varying experience with smartphones. However, we believe that such a test is not relevant until all issues identified in our exploratory usability test have been fixed.

In summary, while we have fulfilled most of our requirements, we still have a number of areas in which we can improve. In the broader scope of *FormAssist*, usability is a key area where we need to make significant improvements and should, thus, be the focus if we were to continue developing *FormAssist*.

## 8.1 Frameworks

In this section, we will discuss a number of choices pertinent to several aspects of *FormAssist*'s development process, namely (a) cross-platform versus native development in general, (b) what the choice of React Native meant, (c) how using TypeScript affected development, and (d) the libraries and models employed for pose estimation.

As stated in chapter 4, *FormAssist* must be developed for both Android and iOS. We, therefore, had to choose a framework that allows us to create a cross-platform application. We spent the first couple of weeks of development prototyping and experimenting with both Flutter and React Native. We identified these as being the best candidates for *FormAssist*, and eventually opted for React Native. Whilst we spent several weeks prototyping—which indeed was more than we initially expected to

spend—we argue that the process, as a whole, was worthwhile because we attained a sound understanding of both frameworks, allowing us to make an informed decision. It is possible that in the context of developing a MVP through a single semester, immediately opting for either framework, and otherwise sticking with it, could have resulted in roughly the same product. However, one of our desired system attributes is extensibility, and when looking beyond our MVP, we are comfortable with our decision of choosing React Native. It should, moreover, be noted that the time it took to adequately prototype both frameworks is directly related to the inherent complexity and novelty of *FormAssist*—applying machine learning to estimate poses using a smartphone—in combination with our inexperience with cross-platform development in general. It is, therefore, possible that our overall experience would have been different had we developed a more generic application without machine learning. Additionally, if we were to develop a similar application in the future, we now know which pitfalls to avoid. Despite the aforementioned hurdles, we, nonetheless, argue that using a cross-platform framework such as React Native is the right choice, especially, when looking beyond a MVP.

Another goal of the project, inherently related to the choice of using a cross-platform framework, was to increase the amount of re-usable code between the two platforms. Whilst we ultimately had to write platform-specific code to adequately support the pose estimation module—as illustrated in fig. 6.14—most of the code in the project was written with an abstraction over platforms using TypeScript. The fact that everything beyond the pose estimation module was developed using TypeScript, additionally, meant that we worked with just a single language—something which meant that only those without prior JavaScript experience had to learn a new language, as opposed to several members having to learn multiple languages.

The majority of our problems with React Native stemmed from the tools required to actually develop an application using React Native in the first place. The dependency management, for instance, relies on a combination of the JavaScript package manager *Node Package Manager* [104] and the platform-specific package managers *Gradle* [105] and *Pod* for Android and iOS, respectively.

The biggest cause of frustration and issues we encountered was, without a doubt, the process of adding new dependencies to *FormAssist*—something which, per the very nature of prototyping and incremental development, happened frequently. Having to deal with three separate package managers, unsurprisingly, proved to be troublesome. Additionally, whilst React Native supports hot reloading, it only worked as intended for Android. Indeed, attempting to debug using hot reloading on iOS caused the application to crash—effectively nullifying the initial benefits of hot reloading. This, inherently, prolonged testing and prototyping for iOS compared to that of Android. Additionally, this issue added to the already slow build times associated with iOS, which could often take two to five minutes. To put this into perspective, build times using Flutter, at the time when we had a preliminary pose estimation application, took approximately 10-20 seconds to build compared to React Native's multiple minutes at a similar stage of development. As we progressed with development, we eventually identified enough issues and worked around them such that the process of actually building *FormAssist* became faster.

As previously mentioned, we relied on TypeScript for the majority of the coding throughout this project. This was a deliberate choice for a number of reasons. Unique to TypeScript is its static type

analysis on compilation which gives a number of guarantees concerning the stability and correctness of the program—most of which are not possible to get using JavaScript. Of course, React Native and the majority of its libraries are written in JavaScript, though this is not usually an issue because the TypeScript compiler instead relies on so-called *type-files*, which are often bundled with the libraries, specifying the types of all data and every function featured in specific library. When a library does not include a type-file, however, it must either be written manually, or forego any guarantees provided by the compiler. Writing a type-file for a large library can be a time consuming task, and thus we opted for the second option. This resulted in TypeScript not providing as much assistance as we had hoped for, though it did ensure our codebase was more robust. The use of 'strict'-mode, in which objects are non-nullable by default, especially aided us in clearly specifying the properties of functions. This, essentially, forced us to handle null-cases from functions where null was a potential return value. In general, TypeScript aided us in ensuring a steady, albeit slower, development process and allowed for easier changes to the codebase.

When pursuing fast pose estimation on smartphones, as per requirement 9 concerning ten frames per second, we tested a number of different machine learning libraries as well as a couple of pre-trained models 6.3. We, initially, used the release-version of TFLite, which employs the CPU for inference, however, it proved to be too slow as per the requirement above. Following this, we tested the library Fritz, however, this also proved to be too slow. Upon further investigation of libraries to test, the Android-only library MACE looked promising, but the integration of it into *FormAssist* was unsuccessful due to dependency issues and build steps with side-effects of yet unknown nature. Eventually, we settled on using a nightly build of TFLite for Android, which supports inference using the GPU, and Core ML for iOS. The whole process of finding and integrating a performant machine learning library into *FormAssist* took much longer than expected. This delay caused further problems in the development process, as other parts of *FormAssist* were reliant on pose estimation being implemented. We spent a lot of time attempting to integrate MACE only to dismiss it, as a result of the inconsistencies we continued to encounter. In retrospect, this might seem like a poor decision, but at the time, it seemed like the best option, given MACE's high performance. Indeed, what we needed then, was a library which performed well, and between the nightly build of TFLite, which we ended up using, and MACE, the latter was by all accounts the most appealing option.

## 8.2 Process Evaluation

In this section, we will focus on the development process used throughout the project as well as the problems we encountered in terms of achieving our goal of developing our MVP. The issues discussed in this section are, therefore, related to the management of the project rather than technical issues. In essence, we have identified the following issues related to our process which we will discuss in greater detail:

- The chosen tools were not properly utilised throughout the project.
- Too many weeks were spent on prototypes; should have made a decision sooner.
- An insufficient amount of integration testing.
- Should we have done cross-platform in the first place? What is gained, what is lost? Does the same apply in the context of a MVP?

For this project, we employed elements known from both scrum and extreme programming development processes. Specifically, we utilised daily stand-up meetings and pair programming. We, furthermore, used a scrum board to keep track of relevant tasks. We chose to only incorporate some of the elements of the processes based on experience with previous projects. We initially used a digital scrum board through ZenHub [106], though we did not have much success with this despite past experience. Generally, we did not take advantage of the scrum board enough and, thus, it was often out of date. We identified this issue halfway through the project, and decided to attempt to fix it. We converted one of our whiteboards to a scrum board. We believed that this could help us be more mindful of the board, as it was always visible in the room. This did remedy the issue at first, but this version of the scrum board was not properly updated either. This was a result of a number of overlapping tasks with unknown time estimations. Additionally, the criteria for the completion of said tasks were vague and, consequently, made it difficult to determine when to move the tasks on the scrum board.

While the prototyping of the pose estimation module was being developed, other group members started working on the cross-platform React Native codebase. This includes GUI, repetition detection, and evaluation modules. In order for the group to work on multiple parts concurrently, we agreed on a set of common interfaces between each module. We, additionally, relied on daily stand-up meetings during project days to keep one another up-to-date with our progress. This approach helped us make continuous progress on *FormAssist*; even when some members encountered problems in other modules. Whilst, we argue, this approach helped maintain a steady development progress, it also meant that some tasks, in hindsight, took longer to complete than they should have. This was because we continued to work on them since no proper deadlines were ever established. An example of this is the repetition detection module which underwent multiple iterations. This was, in part, due to a number of refactoring steps due to the nature of the task; i.e. not knowing which parts of the repetition detection was specific to an exercise and which parts could be generalised. Only after implementing a couple of exercises, was it possible for us to assess which parts could be generalised to encompass all exercises. We, furthermore, arguably, attempted to cover too many edge cases, especially, during early development instead of focusing exclusively on ensuring basic functionality.

We suspect that with a properly defined deadline for the task, we could potentially have avoided, or otherwise minimised, the amount of time it took to complete the module.

Another issue we identified towards the end of the project was the fact that we did not integrate the modules into a single application until late in development process. We, consequently, encountered some issues regrading some edge cases in the predefined interfaces. Again, this was due to the fact that we did not have clear deadlines for the individual modules and the subsequent assembly of them. We suspect this was a result of the amount of times we chose to iterate and improve our model for pose estimation, rather than attempting to assemble *FormAssist* as soon as we had a working prototype, regardless of its performance. Instead, the majority of development time was spent recreating the same functionality with multiple libraries in order to obtain better performance. However, this time was arguably not wasted as we gained significant performance improvements in the latter implementation compared to the first. Nevertheless, this, inevitably, delayed the overall assembly of *FormAssist*.

Testing also fell victim to our relaxed planning and lack of deadlines. Having assembled *FormAssist* late into the project, we could not conduct proper integration testing, which is useful considering the application consists of a number of modules which need to be assembled. On the other hand, unit testing was easier to conduct, due to the fact that each module, and its functionality, can be tested individually and independent of the other modules.

## 8.3 Future Work

This section will outline the potential extensions for *FormAssist* which we imagine will be relevant during continued development. The extensions will go beyond the scope of the MVP.

We have identified two major areas of interest in which we could improve through future development of *FormAssist*:

1. **Motivation**: In chapter 3 we discussed two major factors which affect the course of treatment with a physiotherapist, one being the correctness of the performed exercise, and the other, being the motivational aspect. Whilst we quickly dismissed working on the motivational aspect in this iteration due to time constraints, a larger focus could be allocated to this in future development.
2. **Generalisation**: In chapter 4 we proposed extending *FormAssist* beyond the scope of physiotherapy and transforming the application into an encyclopedia of exercises for everyone to use. This is also a possible extension we could make in future development of *FormAssist*.

### 8.3.1 Motivation

As mentioned in chapter 3, motivation can be achieved in a multitude of ways. We, initially, considered trying to accommodate motivation—more specifically gamification—as part of the MVP for *FormAssist*. This was based on the fact that the physiotherapists we talked to, as well as the conclusions we reached following our research, identified lack of motivation as one of the primary factors for patients neglecting their exercises. However, as we also mentioned in chapter 3, we decided not

to pursue this avenue due to the magnitude of the task and the amount of domain-specific knowledge required. Nevertheless, we believe that gamification would have the most impact on improving *FormAssist* in regards to motivation. The current iteration of *FormAssist* evaluates a user's performance based on a number of factors. Each factor is scored with zero to three stars on each completed repetition. Moreover, it provides an average score for the overall exercise performed. The purpose of this score is to give the user the possibility of, briefly, reviewing their performance in a quantifiable way. As we stated previously, we did not intend on using gamification in the MVP of *FormAssist*. It can, however, be argued that receiving a score for the performance of an exercise is a minor form of gamification in that it may incentivise a user to improve their score, and thus be conscious about their performance. This can, in and of itself, be considered a means of motivating patients to perform their exercises. It moves the focus of the user from performing an exercise, because they have been told to do so by their physiotherapist, to achieving a short term goal. Improving a score is a gratifying experience and thus fits into the first category of gamification as identified in section 3.1.1. It should be noted that this is purely speculation and that thorough testing should be conducted to verify whether the scores, actually, motivate users to continuously perform their exercises, and also improve their performance. For this purpose, A/B testing could be employed by having one group of testers use a version of *FormAssist* where scores are given, and another group of testers use a version where scores are not given. We can collect data about two things; (1) whether they maintain the assigned exercise frequency, and (2) whether their performance improves over time.

In terms of expanding upon the gamification element, we can also explore possibilities relating to the two types of gamification that we identified in section 3.1.1. To reiterate, these options were (1) implementing a reward system on top of an existing activity without altering the activity itself, and (2) altering the activity itself, in order to make it more enjoyable. In the context of future work, we will consider the potential of applying the first type of gamification. The reason for doing this is because adding a reward system on top of *FormAssist* strikes us as being a more appealing option considering how when physiotherapists prescribe exercises to a patient, it is done with a specific goal in mind. If we were to alter the exercise for some motivational gain, we may, inadvertently, remove the ameliorating factor from the exercise and, thereby, essentially nullify the desired effects of performing the exercise in the first place.

As mentioned above, the evaluation scores used in the current version of *FormAssist* can be considered an initial attempt at implementing a reward system. However, this was not the primary intention of the evaluation scores, and the idea can, therefore, be expanded with a dedicated design and implementation. One possible reward system, which has become a staple in many video games, is *achievements*. The idea of achievements, is to obtain badges by meeting specific criteria. Typically, completing an achievement does not grant anything other than being able to display that you were able to complete it. Therefore, the purpose of implementing such a system would be to give a user short term goals which they can aim to achieve. A course of treatment may go on for a significant amount of time without any obvious progress—something which can impede a user's ability to stay motivated. Using short term goals as a milestone can, potentially, distract the user from the fact that their progress may be slow. As an example, an achievement system in *FormAssist* could consist of rewarding a user for achieving the highest score—i.e. three stars—for all repetitions of an exercise or for multiple sessions of an exercise. It would also be relevant to reward a user for performing

exercises consistently. A challenge of making such an achievement system work is that courses of treatment differ from patient to patient and, therefore, the achievements must be generalised such that all users are able to complete any potential achievements. As an example, in terms of creating achievements for rewarding consistency in performing exercises, some patients may have to perform exercises every day while other patients may have to perform them every other day. It would not make sense to implement an achievement for performing exercises every day for an arbitrary amount of days, because not all users may be able to complete it. The achievement system should, therefore, take an exercise plan into consideration. Something which must then also be implemented, as it does not exist in the current version of *FormAssist*.

Another, more direct, approach to motivation are reminders. We have previously mentioned a form of reminder—push notifications. However, this is not the only form of reminders that are at our disposal. Before we move on to some alternative approaches, we will describe our initial approach to reminders and the idea behind it. As we stated earlier, a large majority of the people who neglected their exercises, prescribed from the physiotherapist, listed *forgetfulness* as the primary reason for not doing them. Since we are using a mobile platform the obvious choice was to generate push notifications. To combat the forgetfulness of patients, *FormAssist* can issue notifications reminding them to perform their exercises. Whilst issuing reminders is a relatively simple task, determining *when* to issue them is more complicated. *FormAssist* could be set up to issue push notifications at a specified time, or have *FormAssist* observe the usual times at which a patient performs their exercises and, based on this information, issue a reminder when they have exceeded this timeframe without *FormAssist* registering that they have completed their exercises. Another variation, still using push notifications, is using a technique similar to that which applications like MyFitnessPal use [107]. MyFitnessPal tracks the logins of its users and, if they do not login, issues a notification saying *you have not registered a meal in x days* or *you have x amount of calories left today*. *FormAssist* could employ a similar system, and issue notifications about the user's last login, or how many repetitions or sets they have left of a given exercise, to meet their daily/weekly goal.

Another option we considered during the process of working on *FormAssist* was the ability for physiotherapists to send exercise schedules directly to the patient through *FormAssist*. Additionally, we considered the idea of having *FormAssist* report back to the physiotherapist if and when the patient has performed the exercise(s). The reason for considering this approach was twofold; one being the possibility of extending *FormAssist* beyond physiotherapy, a subject we will discuss in section 8.3.2, and how this, in and of itself, could help improve the forgetfulness of the patients. Additionally, this holds the patients accountable to their physiotherapist due to the fact that the physiotherapist will know if the patient has performed the exercise or not. Furthermore, it could allow the physiotherapist to discuss it at their next consultation. This enforces a sense of responsibility and accountability which serves as a motivational factor by having the patient prove that they are doing their prescribed exercises at the requested frequency. However, this may also have the reverse effect and inadvertently deter the user from using *FormAssist* due to privacy concerns. We must, therefore, remain cautious when considering such a feature for implementation. On the topic of reporting back to the physiotherapist, a possible extension is to have *FormAssist* send the video to the them and, thereby, allow them to view their patient's performance. This means that, in addition to receiving feedback from *FormAssist*, the patient can receive feedback from the physiotherapist. Furthermore, these videos can

be used at their next consultation when discussing their progress. This serves as additional motivation because the patient, and the physiotherapist, can view the progress of the patient over time and see how they improved.

### 8.3.2 Generalisation

A consideration we had in regards to extending the purpose of *FormAssist* was to also encompass fitness. Extending the problem domain raises a number of additional questions in regards to the current state of *FormAssist*. We have to consider the following aspects and extensions in order to determine the viability of creating an application, which extends beyond the realm of physiotherapy:

- Is the current implementation general enough to handle a new problem domain? How much should be changed if it is not?
- Should feedback used for fitness differ from feedback used for physiotherapy?
- Which extensions can be added to *FormAssist* with fitness in mind?

In order to answer the abovementioned questions, we first need to consider which aspects of fitness that *FormAssist* could potentially be viable—considering how fitness is a very broad term. During the conception of the project we, discussed researching whether the concept of *FormAssist* could be made with strength training as a problem domain. This idea was a product of the personal interest of some group members who do strength training themselves. We will, therefore, consider the possibility of using *FormAssist* for strength training purposes as well as physiotherapy.

While physiotherapy and strength training serve different purposes the underlying concept of them are essentially the same. They both consist of performing exercises with the purpose of activating one or more muscles. The difference is essentially the objective of performing the exercises. This means that accommodating strength training may be possible by adding new exercises as is intended by the current design of *FormAssist*. However, an aspect which must be considered is *how* the muscles are activated during an exercise. This is especially important because it is also relevant for evaluating the potential of extending *FormAssist* with more exercises for its current purpose, physiotherapy. The current design is based on the assumption that the target muscle, or group of muscles, of an exercise is activated by repeating a specific movement an arbitrary number of times. An exercise then consists of one or more sets each consisting of some number of repetitions. *FormAssist* currently uses detection of repetitions in order to determine when the user is done performing a set of an exercise. However, some exercises use criteria other than repetitions to determine when they are finished. An example of this is the core exercise "plank". In this exercise a specific position must be held for an arbitrary amount of time. It is possible to consider a set of the plank exercise as consisting of a single repetition. Evaluating the exercise would then be possible using the current implementation of *FormAssist* since evaluations only require that heuristics are defined in order to determine the correctness of a repetition. However, determining when a repetition is done is problematic since it is based on some amount of time passing. This is not possible in the current state of *FormAssist*. The model, as seen in fig. 6.9, would, therefore, need to be updated in order to accommodate this variation. A potential way to do this would be to add a new exercise class as an alternative to the

current one. This class would use a different implementation of repetition detection based on time. The functionality for evaluating performance does not need to change. Exercises such as the plank can inherit from this new class. We, naturally, also have to consider which other types of exercises exist that do not use repetitions as a criterion for determining when a set is done.

In addition to exercise types, we also have to consider whether the feedback provided after a set has been performed should differ depending on the purpose that *FormAssist* is used for. Considering that physiotherapy and strength training serve different purposes, it may be relevant to tailor the feedback to fit the specific purpose. It may even be that different sets of heuristics should be used for the same exercise because it can be performed differently depending on the purpose. As an example the range of motion which is required for an exercise during physiotherapy may be smaller than they would be if performed during strength training. Furthermore, there are different ways to perform a strength training exercises which may not be relevant for physiotherapy. An example of this is the "time under tension" technique. This technique requires specific amounts of time to be used during the positive phase, negative phase, and sticking points of a repetition. Heuristics could be implemented to evaluate whether a user adheres to the technique but they should obviously only be used if the user is performing an exercise with that technique in mind specifically.

In addition to the extensions of *FormAssist* above, an additional feature which could be added is the foundation for coaching. As mentioned previously, we entertained the idea of allowing physiotherapists use *FormAssist* to send training plans to a patient. Much in the same vein, we could extend this feature to allow a coach to send a training plan to a client. With the exercises already in *FormAssist*, the client can be coached in how to perform the exercises correctly, without necessarily requiring the coach to be present. Additionally, as mentioned previously, we could allow clients to submit the videos they record, and send them back to their coach for additional review. Much like with physiotherapists, this allows the coaches to spend more time on designing the workout plan, as well as, ensuring the client performs the exercises correctly after having shown them with the help of *FormAssist*. This extension takes a similar approach to exercise planning and draws inspiration from an application, some of the members of this group use, called Truecoach [108]. Truecoach works by having a coach send out an exercise plan to a client, and whenever an exercise is listed, it attaches a video from its database which instructs the viewer in how to perform the assigned exercise. We imagine *FormAssist* can perform a similar task, however, instead of attaching a video, we have the unique possibility in being able to instruct the client on how to perform the exercise using *FormAssist*'s repetition detection. Additionally, we could also incorporate a video which shows the client how to perform the exercise, should it be an exercise they have never done before. This means *FormAssist* effectively is able to distribute an exercise plan, show how it is performed, and correct/coach the client.

In order to carry out some of the extensions listed above, we would likely need to implement an "account system", meaning patients or clients have to sign up and create an account in order for them to receive their personalised exercise plans. This also entails having an admin or coaching/physiotherapist interface which allows them to have an overview of their clients, create exercise plans, and distribute them to their respective clients. Similarly, one could consider implementing an instant messaging feature which allows the coach/physiotherapist and the client/patient to com-

municate with each other through *FormAssist*. These extensions are arguably more speculative and further into the future than what would be feasible for us to do within an extra iteration of *FormAssist*.

# Chapter 9

# Conclusion

We have designed and developed a cross-platform application called *FormAssist*, which aids patients of physiotherapists in performing their prescribed exercises correctly. The MVP implementation is capable of:

- Estimating a user's pose as they perform an exercise.
- Detecting repetitions of the exercise being performed.
- Evaluating the performance of the exercise based on a set of predefined heuristics.
- Providing real-time, as well as post-performance, feedback.

The exercise heuristics evaluate specific aspects of an exercise. These heuristics have been tested and verified by a physiotherapist, Mr. Jensen. This enables *FormAssist* to provide a score between zero and three stars depending on how well the user performed said exercise. In addition to a star rating, *FormAssist* also provides textual feedback explaining the score and providing suggestions on how to improve that aspect of the exercise. The evaluation is provided after the completion of an exercise, and contains feedback for each repetition performed. Real-time feedback has also been implemented using a sound cue indicating the correctness of a repetition.

*FormAssist* has been designed with extensibility in mind such that new exercises, additional functionality, and even a different pose estimation model can be implemented without major changes to the existing code base. *FormAssist* currently supports leg abductions and bicep-curls. To accommodate portability, we have developed *FormAssist* for both Android and iOS using React Native. Pose estimation has been implemented natively in both operating systems. We utilise Edvard Hua's *PoseEstimationForMobile*-model in conjunction with TFLite beta for Android, and Core ML for iOS. The remainder of *FormAssist* is implemented in TypeScript. We evaluated the usability of *FormAssist* through an exploratory test and discovered various issues relating to repetition detection and evaluation. However, in the context of an MVP, and without more comprehensive testing, the current iteration of *FormAssist*'s GUI is satisfactory.

All functionality defined in our requirements have been implemented. However, we believe issues related to repetition detection and evaluation, should be of high priority during further development in order to achieve an MVP which fulfils all requirements satisfactorily. Despite these issues, we are still confident in the design and implementation of the current state of *FormAssist*.

# Bibliography


[1] Maja Pilgaard Idrættens Analyseinstitut and Steffen Rask. *Danskernes motions- og sportsvaner 2016*. https://www.idan.dk/vidensbank/downloads/danskernes-motions-og-sportsvaner-2016/9a94e44b-4cf5-4fbe-ac89-a696011583d5. 2016.

[2] Daniel Ramskov et al. Maja Pilgaard. *Idrætsskader I Danmark 2016*. http://runsafe.au.dk/fileadmin/user_upload/IDRAETSSKADER_oversigt_revideret_210917.pdf. 2017.

[3] Statista. *Number of members in health and fitness clubs worldwide by region from 2009 to 2017*. https://www.statista.com/statistics/273069/members-of-health-clubs-worldwide-by-region/. 2017.

[4] James G. Garrick and Ralph K. Requa. "Sports and Fitness Activities: The Negative Consequences". In: *Journal of the American Academy of Orthopaedic Surgeons 11* (Nov. 2003). URL: https://insights.ovid.com/pubmed?pmid=14686829.

[5] Department of Health and Australia Human Services State Government of Victoria. *Sports injuries*. https://www.betterhealth.vic.gov.au/health/healthyliving/sports-injuries. 2018.

[6] Direct Orthopedic Care. *Acute and Chronic Sports Injuries*. https://www.directorthocare.com/acute-and-chronic-sports-injuries/. Oct. 2017.

[7] Patienthåndbogen. *Stillesiddende adfærd og helbred*. https://www.sundhed.dk/borger/patienthaandbogen/sundhedsoplysning/overvaegt/stillesiddende-adfaerd-og-helbred/. Aug. 2015.

[8] Videncenter for Arbejdsmiljø. *Stillesiddende arbejde*. https://amid.dk/viden-og-forebyggelse/fysisk-arbejdsmiljoe/smerter-i-muskler-og-led/viden-om-smerter-i-muskler-og-led/stillesiddende-arbejde/. 2018.

[9] Alicia A. et. al. Thorp. "Prolonged sedentary time and physical activity in workplace and non-work contexts: a cross-sectional study of office, customer service and call centre employees". In: *International Journal of Behavioral Nutrition and Physical Activity* (Oct. 2012). DOI: 10.1186/1479-5868-9-128. URL: https://doi.org/10.1186/1479-5868-9-128.

[10] Cormac G. Ryan and Philippa M. Dall et. al. "Sitting patterns at work: objective measurement of adherence to current recommendations". In: *Ergonomics* (Oct. 2011). DOI: 10.1080/00140139.2011.570458. URL: https://doi.org/10.1080/00140139.2011.570458.

[11] Genevieve N. and Dunstan et. al. “Breaks in Sedentary Time”. In: *Diabetes Care* 31.4 (2008), pp. 661–666. DOI: 10.2337/dc07-2046. URL: http://care.diabetesjournals.org/content/31/4/661.

[12] United States Department of Labor. *Ergonomics*. https://www.osha.gov/SLTC/ergonomics/.

[13] U.S. Department of Labor. “Ergonomics: The Study of Work”. In: (2000). URL: http://lobby.la.psu.edu/062_ergonomics_standards/Agency_Activities/OSHA/OSHA_Ergonomics_the_Study_of_Work.pdf.

[14] Vidensråd for Forebyggelse. “Forebyggelse af skader og sygdomme i muskler og led”. In: (2013). URL: http://www.vidensraad.dk/nyhed/skader-og-sygdomme-i-muskler-og-led-er-et-dyrt-og-stigende-problem-i-danmark.

[15] Bruno Bonnechère et al. “Do Patients Perform Their Exercises at Home and why (not)? A Survey on Patients’ Habits during Rehabilitation Exercises”. In: *Ulatas Med J. 2016;2(1):41-46* (2016). DOI: dx.doi.org/10.5455/umj.20160210060312. URL: \url{https://www.ejmanager.com/mnstemps/135/135-1453907032.pdf}.

[16] Wikipedia. *Positive feedback*. https://en.wikipedia.org/wiki/Positive_feedback. 2018.

[17] Juho Hamari, Jonna Koivisto, and Harri Sarsa. “Does Gamification Work? A Literature Review of Empirical Studies on Gamification”. In: Jan. 2014. DOI: 10.1109/HICSS.2014.377.

[18] KhanAcademy. *What are energy points*. https://khanacademy.zendesk.com/hc/en-us/articles/202487710-What-are-energy-points-. 2018.

[19] WeWantToKnow AS. *DragonBox*. https://dragonbox.com/. 2019.

[20] Google. *Earn Health Points to Stay Motivated*. https://support.google.com/fit/answer/7619539. 2019.

[21] Apple. *HealthKit*. https://developer.apple.com/healthkit/. 2019.

[22] Belinda Lange et al. “Development of an interactive rehabilitation game using the Nintendo Wii Fit Balance Board for people with neurological injuries”. In: Jan. 2010.

[23] Valeska Gatica-Rojas et al. “Effectiveness of a Nintendo Wii balance board exercise programme on standing balance of children with cerebral palsy: A randomised clinical trial protocol”. In: *Contemporary Clinical Trials Communications* 6 (2017). DOI: https://doi.org/10.1016/j.conctc.2017.02.008. URL: http://www.sciencedirect.com/science/article/pii/S2451865416300874.

[24] Wikipedia. *Shape Up*. https://en.wikipedia.org/wiki/Shape_Up_(video_game). 2019.

[25] Sangwoo Cho et al. “Development of virtual reality proprioceptive rehabilitation system for stroke patients”. In: *Computer Methods and Programs in Biomedicine* (2014). DOI: https://doi.org/10.1016/j.cmpb.2013.09.006. URL: http://www.sciencedirect.com/science/article/pii/S0169260713003052.

[26] Minyoung Lee et al. “Patient perspectives on virtual reality-based rehabilitation after knee surgery: Importance of level of difficulty”. In: *The Journal of Rehabilitation Research and Development* 53 (Mar. 2016), pp. 239–252. DOI: 10.1682/JRRD.2014.07.0164.

[27] VR Fitness Insider. *Top 15 Best VR Fitness Games - Total Body Workout*. https://www.vrfitnessinsider.com/top-15-best-vr-fitness-games-total-body-workout/. 2018.

[28] Rithmio Edge. “Rithmio Edge”. In: (2019).

[29] Xiaolin Wei, Peizhao Zhang, and Jinxiang Chai. “Accurate Realtime Full-body Motion Capture Using a Single Depth Camera”. In: *ACM Transactions on Graphics* 31 (Nov. 2012). DOI: 10.1145/2366145.2366207.

[30] Mohd Fadzil Abu Hassan, Aini Hussain, and Mohd Asyraf Zulkifley. “Squat exercise abnormality detection by analyzing joint angle for knee osteoarthritis rehabilitation”. In: *Jurnal Teknologi* 77 (July 2015), p. 2015. DOI: 10.11113/jt.v77.6241.

[31] M. U. Islam et al. “Yoga posture recognition by detecting human joint points in real time using microsoft kinect”. In: *2017 IEEE Region 10 Humanitarian Technology Conference (R10-HTC)*. 2017. DOI: 10.1109/R10-HTC.2017.8289047.

[32] Thomas Breinstrup. *Danmark har flest smartphones i hele verden*. https://www.berlingske.dk/virksomheder/danmark-har-flest-smartphones-i-hele-verden. 2017.

[33] Statcounter Global Stats. *Mobile Operating System Market Share Denmark*. http://gs.statcounter.com/os-market-share/mobile/denmark. 2019.

[34] Ayelet Fishbach et al. “How Positive and Negative Feedback Motivate Goal Pursuit”. In: (2010).

[35] et. al. Blundell James Kenne. “The measurement of software design quality”. In: *Annals of Software Engineering* (1997). DOI: 10.1023/A:1018914711050. URL: https://doi.org/10.1023/A:101891471105.

[36] B. W. Boehm, J. R. Brown, and M. Lipow. “Quantitative Evaluation of Software Quality”. In: (1976), pp. 592–605. URL: http://dl.acm.org/citation.cfm?id=800253.807736.

[37] Joseph P. Cavano and James A. McCall. “A Framework for the Measurement of Software Quality”. In: *SIGSOFT Softw. Eng. Notes* (1978), pp. 133–139. URL: http://doi.acm.org/10.1145/953579.811113.

[38] *Mobile OS market share 2018*. URL: https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/.

[39] Steven Chen and Richard Yang. *Pose Trainer: Correcting Exercise Posture using Pose Estimation*. https://www.researchgate.net/publication/324759769_Pose_Trainer_Correcting_Exercise_Posture_using_Pose_Estimation. 2018.

[40] Dan Oved. *Real-time Human Pose Estimation in the Browser with TensorFlow.js*. https://medium.com/tensorflow/real-time-human-pose-estimation-in-the-browser-with-tensorflow-js-7dd0bc881cd5. 2018.

[41] LG. *LG G8 ThinQ Camera 3D Camera and AI Cam for Standout Photos*. https://www.lg.com/us/mobile-phones/g8-thinq/camera. 2019.

[42] OpenPose. *OpenPose*. https://github.com/CMU-Perceptual-Computing-Lab/openpose. 2019.

[43] AlphaPose. *AlphaPose - Real-Time and Accurate Multi-Person Pose Estimation*. https://github.com/MVIG-SJTU/AlphaPose. 2019.

[44] Kaiming He et al. “Mask R-CNN”. In: (2017).

[45] Detectron. *Detectron*. https://github.com/facebookresearch/Detectron. 2019.

[46] Gary E. Mogyorodi. “What Is Requirements-Based Testing?” In: (2003).

[47] Michael Galpin. *Using Web Workers to speed up your mobile web applications*. 2010. URL: https://developer.ibm.com/tutorials/x-html5mobile4/.

[48] SteelKiwi Inc. and SteelKiwi Inc. *Flutter vs React Native vs Xamarin for Cross Platform Development*. Sept. 2018. URL: https://hackernoon.com/flutter-vs-react-native-vs-xamarin-for-cross-platform-development-5f92cfb178ff.

[49] Google. *Flutter FAQ*. URL: https://flutter.io/docs/resources/faq.

[50] *Dart Programming Language*. 2019. URL: http://www.dartlang.org.

[51] Wikipedia. *Ahead-of-time compilation*. https://en.wikipedia.org/wiki/AOT_compiler. 2019.

[52] Martin Bigio. *Introducing Hot Reloading*. https://facebook.github.io/react-native/blog/2016/03/24/introducing-hot-reloading.html. 2016.

[53] Bonnie Eisenman. *Learning React Native*. URL: https://www.oreilly.com/library/view/learning-react-native/9781491929049/ch01.html.

[54] *React*. 2019. URL: https://reactjs.org.

[55] *Open Source Git Projects*. 2018. URL: https://octoverse.github.com/projects.

[56] React Native. *Communication between native and React Native*. https://facebook.github.io/react-native/docs/communication-ios.html. 2019.

[57] Youssef Nader. *React Native vs Flutter: [2019] Everything You Need to Know*. Feb. 2019. URL: https://hackr.io/blog/react-native-vs-flutter.

[58] *Xamarin*. Jan. 2019. URL: https://en.wikipedia.org/wiki/Xamarin.

[59] *Xamarin Development*. 2019. URL: https://visualstudio.microsoft.com/xamarin/.

[60] asb3993. *Introduction to Mobile Development - Xamarin*. URL: https://docs.microsoft.com/en-us/xamarin/cross-platform/get-started/introduction-to-mobile-development.

[61] Microsoft. *ML.NET*. https://dotnet.microsoft.com/apps/machinelearning-ai/ml-dotnet. 2019.

[62] Miguel Deicaza. *TensorFlowSharp*. https://github.com/migueldeicaza/TensorFlowSharp. 2019.

[63] Larry O'Brien. *TensorFlow.Xamarin.Android*. https://github.com/lobrien/TensorFlow.Xamarin.Android. 2017.

[64] Michael Copeland. *What's the Difference Between Deep Learning Training and Inference?* https://blogs.nvidia.com/blog/2016/08/22/difference-deep-learning-training-inference-ai/. 2016.

[65] Khari Johnson. *Google launches TensorFlow Lite developer preview for mobile machine learning*. https://venturebeat.com/2017/11/14/google-launches-tensorflow-lite-developer-preview-for-mobile-machine-learning/. 2017.

[66] Khari Johnson. *Google launches TensorFlow Lite 1.0 for mobile and embedded devices*. https://venturebeat.com/2019/03/06/google-launches-tensorflow-lite-1-0-for-mobile-and-embeddable-devices/. 2019.

[67] Sujith Ravi. *Custom On-Device ML Models with Learn2Compress*. https://ai.googleblog.com/2018/05/custom-on-device-ml-models.html. 2018.

[68] Qian Sha. *Tensorflow Lite for Flutter*. https://pub.dev/packages/tflite. 2019.

[69] React Community. *React Native Camera*. https://github.com/react-native-community/react-native-camera. 2018.

[70] Edvard Hua. *Real-time single person pose estimation for Android and iOS*. https://github.com/edvardHua/PoseEstimationForMobile. 2019.
[71] Fritz Labs. *Fritz Models*. https://github.com/fritzlabs/fritz-models. 2019.
[72] TF-CoreML Contributors. *TensorFlow to CoreML Converter*. https://github.com/tf-coreml/tf-coreml. 2019.
[73] Open Neural Network Exchange. *Convert TensorFlow models to ONNX*. https://github.com/onnx/tensorflow-onnx. 2019.
[74] TensorFlow. *TensorFlow Lite converter*. https://www.tensorflow.org/lite/convert/. 2019.
[75] Towards Data Science. *Deep Learning Framework Power Scores 2018*. https://towardsdatascience.com/deep-learning-framework-power-scores-2018-23607ddf297a. 2018.
[76] TensorFlow. *TensorFlow Lite*. https://www.tensorflow.org/lite/. 2019.
[77] Flutter. *Writing custom platform-specific code*. https://flutter.dev/docs/development/platform-integration/platform-channels. 2018.
[78] Hardik Shah. *React Native Performance: Major issues and insights on improving your app's performance*. https://www.simform.com/react-native-app-performance/. 2017.
[79] Firebase. *ML Kit for Firebase Quickstart, Android*. https://github.com/firebase/quickstart-android/tree/master/mlkit. 2018.
[80] Firebase. *ML Kit for Firebase Quickstart, iOS*. https://github.com/firebase/quickstart-ios/tree/master/mlmodelinterpreter. 2018.
[81] Wikipedia. *Row- and column-major order*. https://en.wikipedia.org/wiki/Row-major_order. 2019.
[82] Wikipedia. *YUV*. https://en.wikipedia.org/wiki/Yuv. 2019.
[83] Xiaomi. *Mace*. https://github.com/XiaoMi/mace. 2019.
[84] MACE. *MACE Dependencies*. https://mace.readthedocs.io/en/latest/installation/env_requirement.html. 2019.
[85] Docker. *Docker Images*. https://docs.docker.com/engine/reference/commandline/images/. 2019.
[86] Edvard Hua. *PoseEstimationForMobile Mace Demo*. https://github.com/edvardHua/PoseEstimationForMobile/tree/master/android_demo/demo_mace. 2019.
[87] Android Developers. *TextureView*. https://developer.android.com/reference/android/view/TextureView. 2017.
[88] Wikipedia. *Kernel density estimation*. https://en.wikipedia.org/wiki/Kernel_density_estimation. 2019.
[89] Vladimir Agafonkin. *Introducing heatmaps in Mapbox GL JS*. https://blog.mapbox.com/introducing-heatmaps-in-mapbox-gl-js-71355ada9e6c?gi=3dde7d536b05. 2017.
[90] Efstathiou Constantinos. *Signal Smoothing Algorithms*. http://195.134.76.37/applets/AppletSmooth/Appl_Smooth2.html.
[91] Google. *Data and file storage overview*. https://developer.android.com/guide/topics/data/data-storage. 2019.
[92] *Realm*. URL: https://realm.io/.

[93] Realm. *What is Realm Platform?* https://docs.realm.io/sync/what-is-realm-platform. 2019.
[94] Facebook. *React Native*. https://facebook.github.io/react-native/. 2019.
[95] Facebook. *React DOM Elements*. https://reactjs.org/docs/dom-elements.html. 2019.
[96] *Resources: CSS properties*. URL: http://www.stylinwithcss.com/resources_css_properties.php.
[97] Facebook. *React Native Basics*. https://facebook.github.io/react-native/docs/tutorial. 2019.
[98] Wikipedia. *Doní repeat yourself*. https://en.wikipedia.org/wiki/Don%27t_repeat_yourself. 2019.
[99] Facebook. *React Component*. https://reactjs.org/docs/react-component.html. 2019.
[100] Facebook. *Jest Testing Framework*. https://jestjs.io/. 2019.
[101] youtube dl. *youtube-dl*. https://github.com/ytdl-org/youtube-dl. 2019.
[102] FFmpeg. *FFmpeg*. https://ffmpeg.org/. 2019.
[103] Jesper Kjeldskov, Mikael B Skov, and Jan Stage. "Instant data analysis: conducting usability evaluations in a day". In: *Proceedings of the third Nordic conference on Human-computer interaction*. ACM. 2004, pp. 233–240.
[104] NPM. *Node Package Manager*. https://www.npmjs.com/. 2019.
[105] Gradle Inc. *Gradle*. https://gradle.org/. 2019.
[106] Zenhub. *Zenhub*. https://zenhub.com/. 2019.
[107] MyFitnessPal. *Fitness starts with what you eat*. https://www.myfitnesspal.com/. 2019.
[108] TrueCoach. *TrueCoach*. https://truecoach.co. 2019.

# Chapter 10

# Appendix

## A Interview Questions

1. Why do people go to physiotherapists in general?
2. How is the course of treatment for your two most frequent types of patients?
3. (a) What differentiates a succesful and unsuccesful course of treatment?

   (b) What are some of the typical problems occuring during a course of treatment?
4. Which tools do you use during a typical course of treatment? (What are the benefits and downsides of each one?)
5. Can you describe a typical day of work?
6. (a) Which injuries do you most frequently see in your patients?

   (b) Which exercises do you prescribe for those types of injuries?

## B Interview 1

Interview af Mads Bomholt Jensen. Interviewer: Jonatan Transcript: Niclas

Jonatan - Velkommen til mødet

Mads - Jo tak

Jonatan - Vi har haft lidt success med at finde andre der har haft held med at kunne detektere hvor ens krops dele var, så det ser lovende ud

Mads - Ja

Jonatan - Og det som vi lige som skal have ud af dig nu er i bund og grund 2 ting. Vi skal have understreget at der er noget i kan bruge det til. Det andet er lidt mere krav specifikt. Hvis vi kan få dig til at pege lidt i en retning af hvad sådan et system skal kunne

Mads - Vi har tidligere afprøvet et system, så vi har nogle ting vi synes det skal opfylde

Jonatan - Fedt, Vi har 6 spørgsmål til dig. Hvorfor kommer folk generelt til fys? Specifikt hvilke skader kommer de med?

Mads - Folk kommer med alverdens skader. Det er typisk fordi de er svækket efter en skade. De kommer jo til mig hvis de f.eks. har fået dårlig ryg, både hvis de har fået dårlig ryg, hvis de er blevet opereret for dårlig ryg, knæ eller hofte. Hvis man har lidt noget funktionstab efter en skade så kommer man oftest til fys. Eller hvis man lider af smerter som nedsætter ens funktionsniveau. Det er selvfølgelig også en grund til at komme til fys.

Jonatan - Og der er ikke nogen som kommer til fys for at forebygge?

Mads - Jo, forebyggelse har vi selvfølgelig også. Det er typisk for at forebygge hjertekarsygdomme og lignende problematikker. Men der er forskel på hvilken type af fys man kommer til: det private, det kommunale eller det regionale. Og jeg arbejder i det kommunale. Så skal man have en henvisning for at komme ind [til fys]. Den får man gennem sygehus eller egen læge. Så kommer man ind under noget lovgivning.

Jonatan - Så det er en bestemt type af patienter I har? Det skal jo så være folk som har været i sundhedsvæsnet?

Mads - Ja lige præcis. Hvis det er inden for det forebyggende område, så kan man godt selv komme og henvende sig. Lovmæssigt der varetager vi: sundhedsloven og serviceloven.

Jonatan - Hvad er der ellers af love?

Mads - Det er de primære vi tager os af [sundhedsloven og serviceloven]. Det dækker meget bredt

Jonatan - Et typisk behandlingsforløb for de mest almindelige patienter. Det er nok to spørgsmål: Hvad er de mest almindelige typer af patienter? Hvad er det typiske behandlingsforløb for dem?

Mads - Vi har f.eks. typiske borgere som har fået udskiftet et led, nyt knæ eller hofte, fordi det har været slidt ned. Så kommer de typisk ind til operation, og efter operationen kommer de ind til os. Så ser vi dem typisk 3 - 4 gange individuelt og så har vi dem på et hold i 6 uger hvor de kommer og træner 2 gange om ugen. Herefter skal de som regel bare varetage deres genoptræning selv og her vile det jo være meget relevant med en app-løsning eller device der kunne hjælpe borgeren videre.

Jonatan - Er kontakten mellem jer og patienten så stoppet ind til at problemet opstår igen?

Mads - Ja, det er lidt forskelligt hvordan man vælger at gøre det. Man kan f.eks. lave en kontrol af borgere en måneds tid efter forløbet er stoppet. Der kunne man godt have kontakten over en app. Et andet behandlingsforløb kan være at en borger har været inde at træne i et behandlingsforløb og

så fortsætter de der hjemme. Her kan man f.eks. mødes en gang hver 14. Dag. Der kunne de godt benytte noget andet i mellemtiden til hjælp med træningen.

Jonatan - Hvad adskiller et forløb der går godt og et forløb som går dårligt? Hvad ser i som hyppige grunde til at det ikke går så godt?

Mads - Hvis borgeren ikke synes det går så godt, så er det typisk på grund af smerter. Det er jo ganske normalt. Det kan komme af at de er opereret. Det kan også komme af at de laver for meget eller de laver for lidt fordi de ikke bruger det led de har en relateret skade i.

Jonatan - Hvad med øvelser der ikke bliver udført korrekt?

Mads - Ja det ser vi ganske tit. Vi har tit borger til at lave øvelser der hjemme og når de så kommer og skal vise os hvad de har lavet, så er det noget helt andet end de vi viste dem. Kvaliteten af udførelsen kan godt svinge meget.

Jonatan - Kan I se at det har indflydelse på behandlingsforløbet? Kan de ligefrem forværre tingene?

Mads - Nej, det er måske for meget at sige at det forværre tingene men det er selvfølgelig ikke så hensigtsmæssigt ikke at bevæge sig korrekt. Det afhænger mere af hvor meget eller hvor lidt de laver. Det er der hvor der skal balanceres.

Jonatan - Hvis vi kigger på hvordan det påvirker jer, at der er problemer i et behandlingsforløb. Hvordan ændrer det så jeres arbejdsgang?

Mads - Det kan typisk være at vi skal se patienten oftere. Det koster jo noget kroner og øre mæssigt. Så hvis det ikke går så godt, så ser vi dem lidt oftere, og hvis det hele bare går snorlige så ser vi dem ikke så meget. Så kan de jo godt varetage sig selv.

Jonatan - Hvis vi kigger på de maskiner og udstyr som i bruger f.eks. til overvågning. Måske nogle af de ting som borger skal bruge der hjemme. Kan du fortælle lidt om det?

Mads - Hvis f.eks. en borger skal lave nogle øvelser der hjemme, så har vi et licens til ExtraLife som er et stort øvelses bibliotek. Det kan vi give med hjem på papir form. Vi kan også sende det på mail. Så kan de kigge på øvelserne og se video af dem. På papirerne er der nogle QR koder så man kan se videoerne. Så det er meget brugbart til at vise hvilke øvelser brugeren skal lave. Redskabsmæssigt så er det typisk elastikker eller sådan noget som bliver givet med hjem. Hvis de skal bruge større udstyr som maskiner så har vi dem typisk til at træne inde hos os i vores center. Medmindre vi mener at de selv kan varetage det i et fitness center. Vi giver egentlig ikke så meget udstyr med hjem.

Jonatan - Må du sende os en håndfuld af nogle af de øvelser?

Mads - Ja da. Det er jo et relativt godt redskab til hurtigt at vise borgeren hvad der skal ske.

Jonatan - Hvordan ser en typisk hverdag ud for dig? Går al din til med patienter?

Mads - Generelt er det meget varieret om jeg har nogle individuelle patienter med forskellige skader. Jeg har måske også 2 timers hold træning med forskellige hold. Så har jeg en masse individuelle patienter ind i mellem [holdene]. Så har man også noget administrativ tid, til at dokumentere det vi

laver, som vi skal i følge lovgivningen. Eller man kan bruge den tid til f.eks. at lave trænings programmer. Man er selvtilrettelæggende. Det er individuelt hvor og hvornår man laver noget.

Jonatan - Med hensyn til nogle øvelser vi kunne smide i den her løsning. Hvilke skader behandler i hyppigst for og hvilke skader anbefaler i typisk til dem?

Mads - Typisk har vi en masse borgere som er blevet opereret og har været inaktive over en længere periode. Det har jo betydet noget tab af muskel og det skal trænes op. Men det kan jo være alt for en ryg til et ben. Så det er et meget bredt udvalg af øvelser. Så det er meget svært at scalere det ned til nogle enkelte øvelser.

Jonatan - Kan du ikke lige udvælge nogle øvelser og skrive lidt om dem når du sender os en mail? Og måske lige skrive hvad øvelsen er godt for?

Mads - Jo. De her øvelser fra ExtraLife har en anatomi figur som viser hvilket område man træner med en bestemt øvelse.

Jonatan - Kan du beskrive hvad en app skulle kunne gøre generelt for at være ideel for jer?

Mads - Jo. Vi har faktisk lige testet en app af. Vi har haft en sundhedsplatform ind over (Kunne ikke lige hører navnet). Vi ville jo gerne have en app som visuelt kunne vise patienterne hvad de skulle lave [øvelser]. Men også nogle push beskeder til brugeren så man kan holde dem i gang. Så push beskeder en en vigtig ting synes vi. Fordi hvis det bare er et trænings program som bare ligger på bordet, så glemmer man det hurtigt.

Mads - En funktion så jeg kan se hvornår de har lavet øvelser. Det kunne være et flueben hvor man vinger af at man har lavet en øvelse. Så jeg har en ide om hvor meget der egentlig bliver lavet. Det er jo også lidt for at få brugeren til at udføre øvelserne i stedet for bare at sige at de har.

Jonatan - Vi fokuserer jo på at kunne se om brugeren har udført øvelsen korrekt, så der kunne vi godt tilføje en funktion som viser om de har lavet en øvelse eller ej. Og det behøvede ikke at være et flueben, som man kan kunne vinge af. Altså at fluebenet først blev sendt til dig når de rent faktisk havde lavet øvelsen.

Mads - Vi har tidligere haft en anden app som kommunikations værktøj, hvor man har kunne fortælle hvordan det har været. Hvor man havde et lille kommentarfelt. Så der kunne man jo kommunikere selvom man ikke havde borgeren.

Jonatan - Hvad tænker du omkring det hele [vores ide]? Vi tænker jo at vi tager nogle øvelser f.eks. et squat

Mads - Det er jo relevant at man kan vælge fra en masse øvelser. F.eks. har vi tidligere haft brugt ExtraLife i forbindelse med den anden app. ExtraLife har ca. 20.000 øvelser. Så det er jo vigtigt at man har de øvelser man skal bruge. Så udvalget er bredt. Jeg ved godt at i ikke kan lave et bredt udvalg.

Jonatan - Vi tænker at vi kan tilbyde et værktøj til jer fys'er så i kan generere de her ting som patienterne skal udføre. I den ideelle verden, så vil vi gerne bare give en video som man f.eks. allerede har som input til vores system. Altså hvis vi kan få en masse gode videoer så kan vi bruge

dem i stedet for at skulle lave en masse data selv. Der kan man jo så bruge jer til at sige præcis hvordan en øvelse skal udføres.

Mads - Altså du tænker at jeg skule optage en video hvordan øvelsen skulle udføres?

Jonatan - Ja altså det ville være en metode. Du kunne også give den et link til den her video som i bruger hos ExtraLife. Baseret på den og brugerens udførelse så ville vi kunne give en smiley alt efter hvor godt øvelse var udført.

Mads - Jeg er ikke helt sikker på hvad du mente der

Jonatan - I bedste Just Dance stil hvor man får en score ud fra hvor godt man dansede, så vil vi gerne give en score ud fra hvor godt man udførte den her aktivitet. Hvis i har 20.000 trænings videoer liggende, så kan vi i den ideelle verden bruge dem til at fortælle hvor godt en patient udføre øvelsen.

Mads - Ja. Så kan borgeren vel heller ikke snyde med øvelsen?

Jonatan - Nej præcis. Så det bliver ligsom også et overvågningsværktøj for jer, så i kan være sikker på at øvelserne er blevet udført.

Mads - Det kunne selvfølgelige være relevant i forhold til at vurdere om det der er blevet lavet. Sådan helt rettighedsmæssigt så ved jeg ikke om i må bruge de videoer der.

Jonatan - Nej. I første omgang så er det også bare en studie projekt, så det bliver noget med at vi skal bruge nogle få videoer og dem kan vi lige så godt selv lave. Men vi skal jo have en måde at kunne skalere konceptet og det kunne jo f.eks. gøres med andres videoer.

Mads - Ja det er rigtigt. Man skal jo vise at konceptet virker. Men det lyder spændende.

# C Interview 2

Interview af: Lars Henrik Dahlberg Interviewer: Joachim Valdemar Yde Transkribering: Peter Vergerakis

[0:00:54] Joachim - Vi har lige en håndfuld spørgsmål til sådan at skabe et generelt overblik over, hvad det her er for et domæne og så gå en lille smule mere specifikt bagefter [sic]. Så vi vil egentlig gerne starte med helt overordnet så høre hvorfor folk generelt kommer til fysioterapeut [sic].

[0:01:11] Lars - Okay. Jamen altså folk de kommer generelt til fysioterapeut med en henvisning. Altså klinikker er jo så forskelligt opdelt, og vores klinik den har så den der henvisningsbaseret [sic]. Så det vil sige at folk går til lægen ... lægen giver dem en henvisning fordi de mener det. Og så tager de den henvisning, den lægger man så ud på noget, der hedder * ud på nettet. Så kan folk selv vælge, hvor de vil hen og være. Når de så kommer op til os eller ringer til os *** og så spørger vi jo altid om de har en henvisning, og hvis de har det, så trækker vi den hjem. Og så får vi den sat i gang, og det vil sige så *** så får de en sygesikringsandel. Det vil sige, de får det lidt billigere. Man kan også komme uden henvisning. Så er det noget dyrere. Altså fordelen ved at have en henvisning, det er jo også, de har lægen i baghånden og [pause] og på den måde er der to med, ikke?

[0:02:26] Lars - Lige for at supplere det, så er det jo det sidste nye fremskridt *** har fremsagt, at lægerne ikke skal udskrive henvisninger længere, at patienter selv skal gå direkte til fysioterapeuten. Så det er det sidste nye foreslag, der er fremme. Højst sandsynligt fordi de nok ønsker, at der skal spares penge. *** en fysioterapeut er nok billigere end en læge. Så et eller andet sted kan de se en fordel ved det, og mange af de der henvisninger vi får, der står jo ingenting på. Lægerne ved ikke reelt noget om * som sådan. Der er specialister, der gør det selvfølgelig, men det er nok igen del af det. Det er nok sådan det hænger sammen. Det ville så blive en udfordring for os, hvis det sker, men altså det er kun sådan et foreslag, der er blevet taget op. Det var bare lige for at supplere med det.

[0:03:20] Lars - Ellers så kommer de på grund af det der med, at de har jo enten kontoransatte folk, som har nakke/skulder problemer eller ondt i lænden, som er nok den typiske...

[0:03:33] Joachim - Jamen, hvad er de mest almindelige typer patienter I så får henvist?

[0:03:38] Lars - Jamen, den mest almindelige er jo lænd og ryg smerter eller nakke/skulder problemer. Altså når det er skulder, det kan være selve skulderen og så ud i armen, altså armproblemer eller skulderproblemer eller nakkeproblemer. Det er nok de hyppigste.

[0:04:01] Joachim - Hvad er et typisk behandlingsforløb for de typiske der så?

[0:04:06] Lars - Jamen, det første man gør, når man kommer, så får man en udvidet tid. Det vil sige man kommer og får en tid, hvor man bliver undersøgt. Hvor man laver en screening af patienten eller personen, og så undersøger man den og finder frem til... altså lige gyldigt om man får en henvisning fra lægen og hvad der står en masse ting [sic], så får man altid *** at gå igennem en screening eller en undersøgelse og præciserer problemet. Og mange gange oplever vi, at der bliver skrevet et, men det viser sig, at det er noget andet, at det så drejer sig om.

[0:04:42] Joachim - Okay, så den første det er måske mere sådan en samtale med patienten for lige at finde ud af, hvad det egentlig drejer som og så tage den derfra.

[0:04:51] Lars - Det kalder vi en *, og det er det første man ellers gør, det er at man lige snakker med patienten og finder ud af, hvad er det egentlig du kommer med. Og så spørger man skarpt ind til, hvor det er henne, og der dvæler man ikke for længe, men man spørger skarpt ind til det og finder ud af, hvad er egentlig problemet. Så går man så i gang med sin undersøgelse bagefter.

[0:05:14] Joachim - Lad os sige, vi har en med skulderproblemer, der kommer ind. Hvordan forløber sådan et behandlingsforløb så?

[0:05:25] Lars - Så vil man som sagt lave den her undersøgelse første gang. Den undersøgelse kan godt strække sig over flere gange faktisk. Altså man kan have brug for at kommer videre med det. Ellers vil man - hvis man lige nævner det helt skarpt - så er det typisk sådan en undersøgelse sætter noget i gang... undersøger patienten og finder ud af hvad problemet er med den her skulder, og så vil man langsomt ligesom prøve at få behandlet det her problem ved at gøre det - som sagt konservativt. Der kan være mange faktorer ind over det her. Det kommer an på hvad for en person der kommer ind. Hvis det er en sportsperson - ung mand eller ung kvinde - der kommer ind og dyrker sport og hård idræt, jamen så er vi jo over i den her boldgade og kigger på det her problem. Altså

skulderproblemer kan være mange ting. Det kan være noget, der er *, altså noget der er dårligt til at holde, så er det jo noget man træner op og arbejder med. Der kan være mange ting. Det kan være noget holdningsbaseret også. Det vil sige, at det kan være noget problem [sic], man har fået, fordi man arbejder i nogle bestemte stillinger rigtig længe, og så begynder skulderen at reagere på det, hvis man gør noget rigtig meget. Så bliver man rigtig god til det. Det vil sige, at man oplever, at kroppen former sig efter den arbejdsfunktion, og derved vil man få sådan en ubalance over år - over tid, og så bliver man meget meget stærk i nogle baner og måske noget svag i nogle andre, og så vil tingene udvikle sig, og så pludselig, så opstår der et problem. Så det kan være sådan den vej ***. Så jeg vil sige, det er sådan lidt komplekst; det kan skyldes mange ting.

[0:07:19] Joachim - Kan du beskrive hvad adskiller et succesfuldt behandlingsforløb fra et som er mindre succesfuldt?

[0:07:33] Lars - Et succesfuldt behandlingsforløb er jo hvor man siger, det her det kan vi jo hjælpe dig med. Og så tror man måske, så kan vi hjælpe dig med det, og så vil man sætte noget behandling igang, og typisk vil kan være noget hjælp til selvhjælp, altså det * jo altid ud i, at vedkommende selv vil være i stand til at kunne fortsætte tingene. Det kan være et forløb der går over en gang. Første gang man ser dem - får informationer, får tingene sat i værk, og så er vedkommende egentlig videre med det. Det kan også være det strækker sig over tre gange, eller det bliver mere kompliceret, så det er noget som er mere vedvarende og derved noget man skal arbejde med over tid. Det vil sige, det kan trække ud og blive et forløb man følger over måneder eller år, men det kan måske være sådan, at man sætter typisk noget i gang, som man selv skal arbejde med, fordi det er det det drejer som om, at man selv skal forbedre det her problem, og så vil man typisk følge vedkommende og ligesom guide videre i at få rettet de her ting til og få gjort de rigtige ting - de rigtige øvelser og arbejde med de her dårlige vaner eller måske noget, man ikke har gjort før og så få det implanteret i hverdagen, så der er også meget coaching i det her. Det kan være et godt forløb og, det er selvfølgelig et godt forløb i det øjeblik det så munder ud i, at vedkommende får det bedre. Vi kan oplevet det modsatte at selvfølgelig alle intentioner kører, men det viser sig, det er mere kompliceret end som så, og så bliver det jo så noget med, at nu skal vi tage nogle andre baner - nu skal vi en anden vej, og så går man ind og siger: ved du hvad, du skal tilbage til lægen eller vi skal have fat i en specialist eller du skal have lavet en scanning, eller noget, og så går man den vej omkring det. Det var bare et eksempel.

[0:09:30] Joachim - Det lød som om, at de har sådan lidt tre typer patientgrupper, altså dem som bare er en enkelt konsultation og så et par gange, og så der, hvor der egentlig bliver lavet et reelt forløb. Er det korrekt forstået?

[0:09:44] Lars - Det kan være mange forskellige ting. Det er bare for at stille et spektre der kan ligge inde for *. *** der kan være nogen, som har brug for noget forebyggende - sådan ligesom sige: Inden det bliver rigtig galt, så kan du gøre det her, men lynhurtigt kan vi løse problemet og sige nu skal du gå hjem og fortsætte de her ting, og så er man jo allerede videre. Så har vedkommende fået noget information og ligesom fået rettet sit problem derhen hvor man kan løse det selv [pause] og det er jo en hurtig [sic]. Og så er der dem hvor man lige bliver nødt til at følge op på det, og det er klassisk sådan en god ting at gøre, at man lige sådan siger: Hvordan har det gået? [sic] - Har du kunnet gøre tingene rigtigt? Og så ligesom få justeret det ind. Og så kan det vise sig at være nogle få gange - tre til fem gange, måske, det kan også være mere. Og så kan der også det forløb [sic], hvor man siger:

Det her, det er et langt forløb, som vi bliver nødt til at træne. Det skal siges, vi har meget af det her lange forløb, hvor man sådan følger dem, hvor de bruger nogle træninger, hvor vi sådan ligesom guider dem ind i nogle øvelser, og så træner de jo selv.

[0:10:54] Joachim - Hvilke udfordringer - hvis nogen - oplever i ved den type behandlinger [sic], hvor i har et længere forløb? Altså er der udfordringer der? For eksempel, er der nogle typiske problemer, der opstår - alt lige fra udførsel af diverse opgaver, eller folk glemmer det, eller andet.

[0:11:23] Lars - Jamen det der er kunsten, det er at holde kedlen varm - motivere vedkommende sådan de kan se fornuften i at lave de her ting, og de kan mærke at det gør en forskel for dem. Det vil sige at det er meget vigtigt, for at de bevarer den gejst, og de kan se ideen i at fortsætte den her træning, så har de brug for at få nogle success oplevelser. Det er vigtigt for os hele tiden at *** og ligesom behandling sådan at de kan mærke, det gør en forskel for dem [sic]. Det er der udmunder sig i en success - det er at patienten kan mærke, at det her det er noget jeg kan mærke at jeg kan blive bedre [sic], hvis jeg gør de her ting, og jeg har også en fornemmelse af det. Så holder folk ved. Og det er det det drejer sig om. Problemet er netop hvis der ikke er en klar overenstemmelse og forventningerne er forskellige. Hvis man kommer som kunde eller som en person og skal have ordnet det her problem. Og de har jo nogen forventninger, og som behandler sætter vi nogle andre forventninger. Vi er ikke afklaret omkring det. Så det vil sige, det ligger meget hos fysioterapeuten at man sørger for at afstemme de forventninger, sådan at man er klar i mælet - sørge for at fortælle, hvad er det det her drejer sig om, og ligesom at man ved: Det her det kan jeg give dig en klar besked om, og sige: Du skal ikke forvente måske det her det bliver 100 procent *, det kan blive måske 80 procent, men det må vi prøve at arbejde videre med, for eksempel. Eller sige: Der er god sandsynlighed for, at det kan blive godt. Og så skal man hele tiden vise dem vejen og ikke sætte nogen forhåbninger som ikke overholdes, så der er rigtig meget pædagogik i det her også. Som behandler har man jo rimelig styr på sin viden omkring det.

[0:13:30] Joachim - Ja, så det virker til det med at dels motivere folk og holde dem motiveret er faktisk kilden til et godt behandlingsforløb.

[0:13:40] Lars - Ja, altså det er jo en ting at holde dem motiveret og så sørge for, at de hele tiden mærker, at de sådan synes, hertil kan jeg mærke, og så ligesom hele tiden være på forkant, så man tager dem ligesom på forkant, frem for at de skal komme og melde ud med det, så siger man: Jamen jeg kan mærke sådan og sådan, og hvor er du henne? Man spørger meget, man snakker meget. Det er meget coaching - rigtig meget coaching, kan man sige. Man kan sige ***, men det der med at man hele tiden er afklaret, og at man også sørger for, at hvis man ikke kan hjælpe dem mere, at man også siger fra, det er lige så vigtigt. Altså man som terapeut siger: Jeg kan hjælpe dig hertil, nu kan jeg ikke hjælpe dig mere, men jeg vil godt *** hjælpe dig til at komme videre. For eksempel synes jeg at du skal gå derhen, for eksempel. *** og så kan det være godt. *** og så få ordnet ved speciallægen, for eksempel.

[0:14:40] Lars - Ja, det kunne være sådan et eksempel på, hvor man også ligesom fortæller den her person: Jeg kan hjælpe dig med det og dertil... hvis du skal videre med det, så tror jeg du har brug at komme et andet sted hen. Det kunne være i ret mange sammenhænge. Det kunne være, ja... en kompetence som man ikke har, og så kan man så viderebringe den... til personen et andet sted

hen.

[0:15:07] Joachim - Hvis vi tager et længere behandlingsforløb, hvad værktøjer benytter i så typisk i forbindelse med det? [sic]

[0:15:18] Lars - Med et længere behandlingsforløb. Jamen, som sagt er det meget med at have en plan. Og jo mere skitseret den er, jo bedre er det. Altså det bliver mere og mere sådan. Altså der er flere og flere sådan udskårne. Der er lavet nogle forskellige forskningskoncepter på forskellige behandlinger, for eksempel akillesseneproblematiker eller hælproblematiker, for eksempel ***. Og der er der lavet sådan nogle forskningskoncepter på det, og man har sådan noget træningsbaseret og øvelsesbaseret med *, som man har forsket på, og de der forskellige forløb kan være rigtig fine at gå ind og benytte sig af, fordi der har man sådan noget man ligesom ligger frem, og så siger over så lang tid kan man se, og ud fra de undersøgelser man viser, kan man se, så er sandsynligheden for, at hvis du bruger det her tre gange om ugen, så mange gentagelser om dagen, sådan. Hvis du gør det, jamen så er der stor sandsynlighed for, at du kan komme langt med det her, for så følger du en linie, som er lidt forskningsbaseret, og det vil sige, at det er godt. Så er der også nogle ting du skal holde dig fra, så man giver en rigtig meget god information [sic], og noget de kan holde sig til lidt sådan bam bam. Det er ikke sådan noget løst noget. Det er ikke til at misforstå. Det er sådan ligesom konkret. ***. Lad os snakke sammen, når du har gjort ***. Så du kan gå ud fra noget. Du har konkret materiale du kan arbejde ud fra. Så kan du også som terapeut også være meget mere - kan man sige - skarp, og du kan også bedre ligesom følge op på nogle ting, og så kan du holde de her mennesker op på nogle ting. Har du lavet dit arbejde? For eksempel, ikke? Eller: hvordan har det stået på i sidste uge? Og så kan du spørge meget mere konkret ind til det her, for at få det her program til at fungere. ***.

[0:17:27] Joachim - Hvor tit oplever I at folk ikke får lavet noget? Nu siger du at du spørger ind til om de har fået lavet det.

[0:17:34] Lars - Jamen, jeg oplever det tit at, der er mange ting der kan... Så er vedkommende syg eller, så har der været, ja... udsat for et eller andet [sic], der gør at man har... familiesammenkomst, så har man ikke fået gjort så meget, og det er jo selvfølgelig fair nok, så tager man den jo... så op på hesten igen, og så man tager den jo der. Så er det jo vigtigt, at man ikke slår folk oven i hovedet med det, men man siger: Ved du hvad? Så arbejder vi ud fra det du har, og så kunne sige: Hvis vedkommende bliver ved med at komme at sige: Jamen jeg har ikke... Jamen helt ærligt, så tror jeg, du skal lige tænke over det, når du er klar så komme tilbage, når du synes du er klar. For eksempel ikke, vi skal ligesom... også åndssvagt at bruge folks tid og penge og ressourcer og vores ressourcer på noget, der ikke måske er en success så godt [sic], og så er det vigtigt at man melder klart ud, så det giver jo det bedste - kan man sige - de bedste resultater i sidste ende, hvis man kan melde klart ud med nogle ting, ikke? ***

[0:18:39] Joachim - Det lyder lidt til at man får sådan skræddersyet et skema, som passer til lige præcis den skade du har, men er det rigtigt forstået at du så nærmest tager udgangspunkt i en række standard øvelser, og så bliver det så mængden og antal gentagelser og så videre af alt efter hvad personen nu lige døjer med?

[0:19:03] Lars - For eksempel sådan en situation, er det meget... Der er lavet - for lige at nævne den - er der jo for eksempel, noget der hedder: Glad i dag, sådan glad træning, som er landsbaseret.

Det er til folk med gigt i tæer of hofter for eksempel, og nu er det også kommet til ryg. Og det er sådan landsdækkende noget [sic], som man kan få gjort og arbejde med. Så kommer man ind og får en undersøgelse, og så har man en række øvelser, man skal igennem. De øvelser kan man jo så justere lidt på og arbejde ud fra. Men det er et godt udgangspunkt at have. Jeg vil sige generelt er det et godt udgangspunkt at have det her materiale, fordi der har siddet nogen og arbejdet ud fra noget grundlæggende for - lad os sige - knæene. Og så alle de ting, det er fint, så har man noget grundlæggende der, og så ved man ud fra de øvelser der, der har vi noget - kan man sige - undersøgelsesmateriale over det. Der er brugt masse tid på det, der er en masse mennesker der har været igennem det, så man har undersøgt det. Og så har man jo selvfølgelig en hvis viden ud over det, så siger man... Som terapeut er det så vigtigt at man sådan ligesom husker, at det er en individuel ting, det vil sige, at det er en person der kommer, og vedkommende har sine ting, så det skal man tage højde for, og for at man så får lagt et program der passer til vedkommende, så har man pludselig nogle flere strenge at spille på. Du har noget videnskabsbaseret, du har din egen læring, som du også din egen viden [sic], og du har også den her samtale med personen, så du kan jo pludselig skræddersy det, måske, så effektivt som muligt, så det er ikke bare sådan du får... Det er ikke sådan man kunne være en receptionist og bare kunne komme: Kan du ikke lige lave mit arbejde? Jo, jeg printer bare lige det her program ud. Sådan hænger det jo ikke sammen. Men det ville være let, hvis det var sådan. Det er bare ligesom et materiale, som gør tingene meget mere validt.

[0:21:12] Joachim - Når første gang patienten bliver introduceret for de øvelser her, de skal lave derhjemme, bliver det så vist på... Hvordan foregår den proces?

[0:21:24] Lars - Nåh jamen så gennemgår man tingene sammen, så er man sammen med vedkommende, og så viser man øvelserne sammen, og så går man øvelserne igennem, og så træner man typisk dem, og så retter man, fordi det der med at kunne justere sit knæ og stå rigtigt og have den der kropsfornemmelse, den er altså * værd for mange, og det er der det bliver meget udfordrende, for det kan godt være man siger det. Du skal have knæet... prøv at holde det der, ***. Og så skal man lige overføre den bevægelse til hovedet. Det er en ting at få det at vide, noget andet er at overføre det til kroppen, så den der *, kan man sige, den kan ligge på et lille sted. Og det er meget lettere, hvis man har en rimelig høj kropsfornemmelse og en god fornemmelse, hvor man skal være henne, så er det jo ikke noget problem, men det der er det store problem ved os, hvis vi ikke er til stede, så bliver tingene pludselig et problem, fordi de gør det forkert, og det er det store arbejde vi har, ikke? Det er derfor de kommer gentagende gange, fordi nogen mennesker har ikke den kropsfornemmelse og overføre bevægelse... teori til praksis.

[0:22:50] Joachim - Så det at huske, hvordan de skal lave dem ordentligt...

[0:22:55] Lars - Ja, og mange de kommer jo fra et arbejde, hvor de bruger hovedet helt vildt. Det er jo hovedet det hele, så pludselig skal man til at overføre... De har siddet og arbejdet med materiale hele dagen i en helt anden verden, så kommer du efter arbejde måske, kommer ud... Hvad var det nu jeg skulle? Man bliver god til det man gør, men hvis man bruger hele dagen på noget andet, så kan man ikke forvente at man kan blive verdensmester. Så derfor er det vigtigt, at man ligesom hjælper dem godt på vej. Og det kan enten være rigtig meget, eller nogen der er det bare sådan et lille skub, hvor man bare lige justere lidt. Andre der skal man simpelt hen holde dem i hånden, ikke? Og så skal du kigge dem i øjnene og så skal du holde hænderne på knæet og så *** gøre det gentagende gange,

og det skal du også gøre næste gang. Det er bare svært at overføre det. Så nogen har brug for meget personlig træning, andre kan få det sådan lidt mere løst, ikke? Sådan lige hurtig information, og så kan de selv gå videre med det. Så det er meget forskelligt jo.

[0:24:05] Joachim - Lige for sådan og runde af måske: De problemer du har opridset nu, har du noget i tankerne, som du synes kunne være en ideel løsning? Om det er, at få folk til nemmere og huske hvordan de skal lave øvelser [sic], eller bare det at huske det i sig selv. Er der nogen værktøjer her, som I måske allerede bruger, eller som du synes I mangler?

[0:24:35] Lars - Jamen, der findes jo sådan nogle træningsenheder, hvor man går ind på sådan nogle hjemmesider, så kan man finde øvelser, og så kan man printe dem ud. Man kan også lave sådan en tegning, hvor man selv lige sådan... at man hvis man er god til det, så kan man jo selv tegne det ind og vise det, så får de det i hånden, og så sætter de det op på opslagstavlen derhjemme og så kan de lave det hjemme. Så sådan et pædagogisk stykke redskab, hvor de selv husker det. Aller bedst er det, hvis man kan lægge det ind på rygmarven, og så kan de bare lige bringe det frem, ikke? Og det er derfor gentagende gange er det godt at gemme det og få det her gjort igennem. Så det er det jeg synes der er aller bedst, men ellers så får man jo et stykke materiale med hjem, men altså jeg ved med sådan noget materiale, det kan let komme til at lægge i en eller anden skuffe, eller, nåh ja, så sidder det der, og så er det i virkeligheden svært at overføre det. Men det er typisk sådan et redskab man kan bruge. Jeg bruger meget telefonen også til at optage billederne og * små videosekvenser på deres egen telefon, og så kan så...

[0:25:44] Joachim - Så kan de afspille dem, når de kommer hjem.

[0:25:46] Lars - Ja så kan de... Typisk de ældre mennesker, der er nogen der har... som ikke husker så godt. Korttidshukommelsen, den er ikke så god til... Selvom jeg har sagt det der, da da, og nævnt [sic], så kan de ikke rigtig overføre det og få det gjort. Så derfor kan de være en fordel lige at optage en lille video eller tage nogle billeder af det, så de sådan kan genfinde det. Det synes jeg egentlig er meget praktisk.

[0:26:16] Joachim - Ved du hvad? Vi er super glade for, at du lige ville snakke med os. Vi har fået en masse stof at tage fat i.

[0:26:37] Peter - Du snakker om det her med at det kan være svært at huske, hvordan man laver en øvelse rigtigt og så videre, at nogle patienterne ved sådan noget. Oplever du nogle gange, hvis du nu har givet en patient nogle øvelser med hjem, at deres skade eller problem faktisk ikke er blevet bedre eller måske endda værre, fordi de har måske lavet en øvelse forkert, eller ikke har fået så meget ud af det?

[0:27:03] Lars - Altså ja, men det hjælper jo, så hvis man så har illustreret det ordentligt. Det kan opstå lettere, hvis man bare taler om det, og man synes man rent pædagogisk, man virkelig har forklaret det her helt efter bogen og virkelig sådan fået skåret det ud i pap, ikke? Og så kommer de hjem, og så synes de, de egentlig skulle gøre sådan og sådan og så viser det sig at de har gjort noget helt andet. Det kan jeg godt opleve at folk kan finde på det, og den skal man også lige kunne gennemskue. Er den person sådan en, hvor man ligesom kan sige det nok så pædagogisk, men det viser sig bare, at det hænger ikke ved på samme måde, fordi de har ikke den der, måske kropsintelligens. Så jo, det

genkender jeg også, og derfor er det mange gange materiale der skal til, for at de kan få det gjort, og derfor følger vi også op på det tit. Altså, prøv nu at bruge det, så jeg er sikker på, at de ikke laver noget helt forkert, som du siger. Så får de ikke en skade af det, netop for at undgå det som du siger. Jo, det genkender vi.

[0:28:16] Lars - Det er jo dejligt, hvis man har... Og derfor kan jeg godt bruge en telefon og sådan noget, til dem selv. Det er hurtigt, ikke? Det er meget hurtigt at gøre, og så kan man bruge tiden mere effektivt til den her kunde eller den her person. Fremfor at bruge en masse tid på at printe ud og finde det inde i programmer. Og finde de her øvelser og alverdens, de er jo * udspecificeret, så det kan godt tage lang tid, og jeg kan også sidde og arbejde med det selv, så hvis man lige kunne gøre det, et redskab, hvor man hurtigt kan bruge sin mobilenhed, så er det jo lynhurtigt pædagogisk redskab til vedkommende. Jeg har haft en enkelt - hvis jeg må supplere lidt - jeg har haft enkelt, som siger: Nåh, nu tog du lige et billede på din telefon. Og så viste jeg, jeg sletter det lige igen. Nåh ja, fordi, har du tilsluttet dig for eksempel... Han var bange for at det gik op i skyen, du ved. At det går ud. Selvom at hvis jeg har bedt om og alt mit materiale bliver lagt... [sic] Så det er jeg ikke. Så det er klart, der er også noget der, man skal være opmærksom på selvfølgelig.

[0:29:35] Joachim - Noget med noget privatliv og så videre.

[0:29:38] Lars - Ja, med hensyn til det materiale, altså man nu laver. specielt hvis jeg bruger det på min egen... Dem som er sat ind i tingene og kender det, de er lidt mere opmærksom på sådan nogle ting. Så derfor er det bedst, hvis de gør det på deres egen telefon. Ellers hvis man bare ligesom har det der redskab helt på skarp på det [sic], så kan man jo godt bruge det på sin egen, mener jeg, så skulle det vidst ikke være noget problem. Det ved I mere om.

[0:30:14] Joachim - Lars, lige helt til sidst. Tror du det er muligt, hvis vi kan prøve at se, hvordan sådan et behandlingsforløb ser ud. Nu nævnte du, at du havde nogle øvelser på computeren. Er det noget, som vi kan tilgå, eller vi kan finde et eller andet sted?

[0:30:30] Lars - Jamen, det er jo sådan en enhed man køber. Altså det er jo noget man som fysioterapeuter kan købe. Jeg skal nok prøve at åbne det op og i kan prøve at se det. Det er sådan meget enkelt. Altså, det er sådan lidt, man har sit login, så kommer der en kæmpe stor database med en masse øvelser, hvor man kan gå ind, og hvad hedder det... måske rygøvelser, så kigger man på det, så kommer der et hav af... så kommer der lændeøvelser. Det kan også være rotationer, så kommer der en masse øvelser, der roterer. Så det er sådan meget tegnet, og der er også, hvor det er lavet med billeder, altså rigtige personer. Der er også nogen, hvor de har det med små videosekvenser, der kører, hvor øvelsen vises, så den er sådan meget pædagogisk også. Det er sådan lidt noget omfattende, og man skal ind og bruge det, og det kan være redskab, man kan vise. Selvfølgelig er det det, så jo. Og jeg skal nok prøve at finde navne på, hvad det hedder, hvis du godt kunne tænke dig det, så skal jeg nok lige prøve og...

# D Key points

listing 10.1 shows which 14 key points are returned exactly. The numbers correspond to the index in the vector containing the $x$ and $y$ coordinates for the given key point. This design allows us to access

a given key point by writing `body[RIGHT_SHOULDER]`, for instance. The same key points are present in the iOS code.

```
const HEAD = 0;
const STERNUM = 1;
const RIGHT_SHOULDER = 2;
const RIGHT_ELBOW = 3;
const RIGHT_HAND = 4;
const LEFT_SHOULDER = 5;
const LEFT_ELBOW = 6;
const LEFT_HAND = 7;
const RIGHT_HIP = 8;
const RIGHT_KNEE = 9;
const RIGHT_FOOT = 10;
const LEFT_HIP = 11;
const LEFT_KNEE = 12;
const LEFT_FOOT = 13;
```

**Listing 10.1:** Java code for key points.