Università degli Studi di Padova

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea in Ingegneria dell'Informazione

Libreria Bluetooth in React Native per dispositivo di allenamento muscolare del pavimento pelvico

Relatore:

Prof. Damiano Varagnolo

Correlatrice:

Dott. Roya Doshmanziari

Laureando:

Jaswant Singh Bogan

Anno Accademico 2021-2022

23 Settembre 2022

Ai miei genitori e ai miei fratelli,
che mi hanno sempre sostenuto
Al mio relatore e alla mia correlatrice,
per la loro disponibilità e gentilezza
Ai miei più cari amici,
per il loro supporto

Abstract

The pelvic floor muscles have an important function which is to support organs such as the bladder, bowel and uterus. However, in women, these muscles are weakened or damaged due to many factors such as childbirth or injuries. A way to strengthen these muscles is to carry out exercises, which can be dull or done incorrectly.

An app was created to make these exercises entertainment, the application should also make it easier to track the user's progress over time, and ensure that the exercises are training the pelvic muscles according to public health recommendations. This report shows the team's efforts in creating such an application.

This thesis is the continuation of the project for developing and delivering an adaptable mobile game for pelvic floor muscle training.

The application was requested by Damiano Varagnolo, head of a research group at NTNU and my professor of digital systems. The requested project goal was almost completed by late 2021 and few things remained to implement, one of those was to connect the device with the app through a Bluetooth connection, for this purpose I have created Bluetooth library to connect the pelvic floor muscle training device to the mobile game.
In order to test the library I created a test application, with the same technologies that the group used to create the game.

In conclusion, I documented and videotaped the test application, showing that the data requested by the pressure monitoring device was displayed correctly and responded to user input.

Sommario

I muscoli del pavimento pelvico hanno un'importante funzione che è quella di sostenere organi come la vescica, intestino e utero. Tuttavia, nelle donne, questi muscoli sono indeboliti o danneggiati a causa di molti fattori come parto o lesioni. Un modo per rafforzare questi muscoli è eseguire esercizi, che può essere noioso o fatto in modo errato.

È stata quindi creata un'app per rendere questi esercizi coinvolgenti, l'applicazione dovrebbe anche renderli più semplice per monitorare i progressi dell'utente nel tempo e assicurarsi che gli esercizi stiano allenando il pavimento pelvico secondo le raccomandazioni di salute pubblica. Questo rapporto mostra gli sforzi del team nella creazione una tale applicazione.

Questa tesi è la continuazione del progetto per lo sviluppo e la consegna di un gioco mobile adattabile per l'allenamento muscolare del pavimento pelvico.

L'applicazione è stata richiesta da Damiano Varagnolo, capo di un gruppo di ricerca presso NTNU e mio professore di sistemi e modelli. L'obiettivo progettuale richiesto era stato raggiunto verso la fine del 2021 in però mancavano da implementare alcune funzionalità, una di queste era collegare il dispositivo con l'applicazione tramite una connessione Bluetooth, per questo scopo ho creato una libreria Bluetooth per collegare il dispositivo di allenamento muscolare del pavimento pelvico al gioco mobile.
Allo scopo di testare la libreria ho creato una applicazione di prova, con le stesse tecnologie che il gruppo ha utilizzato per creare il gioco.
In conclusione, ho documentato e videoregistrato l'applicazione di prova, mostrando che i dati richiesti dal dispositivo che monitorava la pressione venivano mostrati correttamente e rispondevano all'input dell'utente.

Contents

List of Figures

Chapter 1

Introduction

The Pelvic Floor Muscles (PFM), are important as it supports the pelvic organs such as the bladder, the intestines, and in females, the uterus (Figure 1.1). If the pelvic floor muscles become damaged or lose their strength, usually due to pregnancy, childbirth, or injuries. This could lead to pelvic floor lapses and 1 in 3 women experience urinary incontinence. These conditions become more prominent in women following menopause. Therefore, to relieve and cure the above issues, women are encouraged to strengthen the PFM, by carrying out the Kegel exercises.

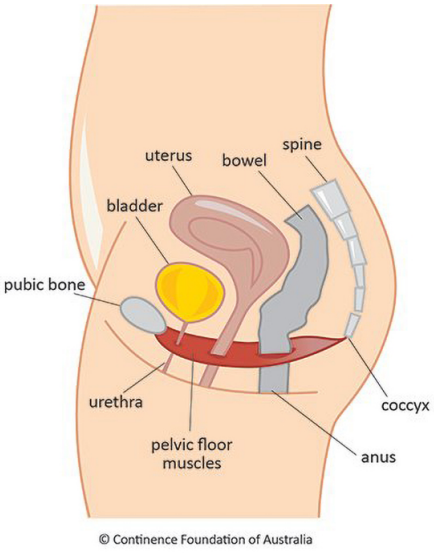

Figure 1.1: Illustration of female pelvic floor - Source: continence.org.au

1.1 Kegel exercising

This exercise is known to be successful in reinforcing the muscles through repeated contractions and relaxations. The Kegel exercises, although helpful, are required to be carried properly and it needs to be repeated over a long period, to fully strengthen the pelvic floor muscles. About 30% of the women perform the exercise incorrectly, this is given to different factors, including the nature of the location of the muscles and the inability to acknowledge the activation of the pelvic floor muscles, or the surrounding pelvic, abdominal, or hip muscles. Most women also tend to neglect or do not exercise sufficiently, this is given due to many reasons and one of them is the lack of feedback that women receive in regards to whether the exercise is performed correctly. In addition, among other factors, female sexual health is considered a taboo subject, therefore women find themselves exercising within the confines of their homes, away from the company.

To tackle this issue, of women performing the Kagel exercise in private without feedback, leading to possible harm to their PFM due to wrong exercises, women are offered to play a game app that uses real data from vaginal pressure sensors.

1.2 Exercises gamification

This app not only provides biofeedback about the performance of the exercise, but it also provides a midterm increase in fitness due to the training outcome and motivates women in a suitable chosen game. Furthermore, it encourages women in exercising by making it fun and engaging, it also increases awareness and destigmatizes female health. The data collected during exercises using the game is not only beneficial for research purposes, but it allows for adapting the game difficulty to the players' current fitness and fatigue status to ensure effectiveness and fair game dynamics.

1.3 Femfit device

The Auckland Bioengineering Institute, University of Auckland (NZ) developed an intravaginal pressure sensor array to better understand the dynamics of the PFM. The prototype known as Femfit (Figure 1.2) it has eight equal-spaced pressured sensors, connected to a flexible printed circuit board (PCB) and encapsulated in a soft medical grade silicone. The Bluetooth communications are transmitted by the telemeter, which is attached to the PCB, enclosed in plastic, which sits outside the body. The Femfit is 80 mm long, 24 mm at its widest point, and 4 mm thick, allowing the device to be flexible and able

to conform to the vaginal anatomy. The sensor array enables the simultaneous measurement of the pressure profile along the length of the vaginal duct. Thus, the sensors at the most distal end of the vaginal canal (sensors 7 and 8) will measure abdominal pressure, while sensors 3 to 6 are most likely to measure the pressures developed by the pelvic floor muscles. The Femfit is thus able to provide feedback on whether the Kegel exercises are being performed correctly (e.g., contracting the PFM and not the abdominal muscles) or not.

Figure 1.2: Photo of a Femfit pressure sensor. Source: aucklandnz.com

The device consists of 8 pressure sensors within a soft medical grade silicon enclosing, connected to a Bluetooth communication module through a dedicated flexible wiring.

1.4 Kegeland application

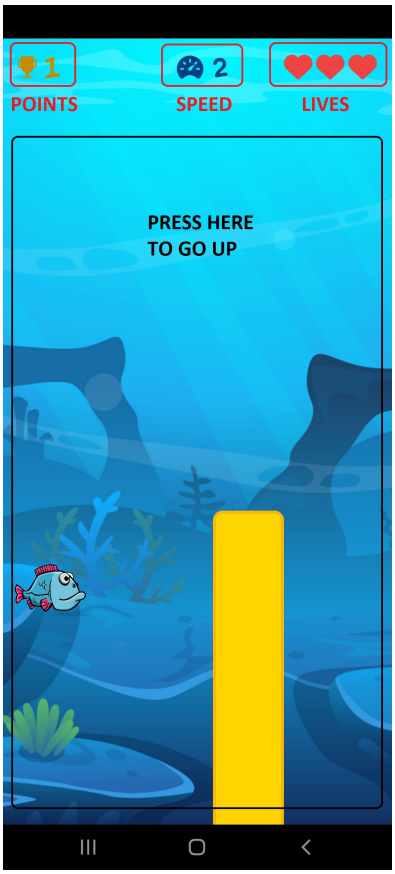

Figure 1.3: In-game screenshot from an exercising session. Source: [3]

Due to many limitations caused, including the application only being accessible on Android devices and the inability to implement functions that were necessary to make the application production ready, a new version has been created removing these limitations, and more features have been incorporated allowing more flexibility and compatibility with different operating systems such as Android and IOS. In addition, by using machine learning algorithms the research group was able to explore the game configuration and this allowed to adapt its difficulty and make it more specific for each player based on their performance, giving the possibility to give both users and physicians information and statistical data about the users' estimated current level of fitness.

The game is a type of running game where the character has to move to avoid obstacles. While playing, the application collects measurement data from the Femfit via Bluetooth. The user wears the Femfit and uses the pressure sensor as a game controller. By contracting the PFMs, the user sets off a jumping action to avoid incoming obstacles.

(See Figure 1.3). When it avoids an obstacle it gains one point and when it hits an obstacle it loses a life. The starting lives are 3 and the game ends when all lives are lost. Lives and points are shown on the top of the screen. The speed of the game, shown as well on the top of the screen, can be changed from 1 up to 5 to increase the difficulty of the game. The aim of the exercises is thus to avoid obstacles. Before each gaming session, the user can set her personal, maximum contraction pressure through a calibration routine. This information is then used to set the threshold pressure, which the measured pressure is required to exceed to actuate a jumping action in the game. The most relevant parameters of each exercise are the required length and frequency of the jumping actions, combined with the number of jumping actions required from the user to finish one course.

1.5 Fatigue modelling problem

The Adaptive Pelvic Floor Muscle Trainer research project developed a piecewise linear dynamical model to study the short-term effects of fatigue and recovery on the strength of pelvic floor muscles [1].
The final model presented by the research group is based from an existing model for skeletal muscle fatigue, proposed in [2], and then adapted because it did not include any driving force or dynamics for muscles to get back into a resting state after activation. For example when playing the game, women's only activate their muscles when intending to jump over an obstacle.

The model presents the following quantities:

- $m_{\mathrm{a}}(t)$:= number of motor units that are in an *active state* at time t, and that are activated by a voluntary drive;
- $m_{\mathrm{f}}(t)$:= number of motor units that are in a *fatigued state* at time t;
- $m_{\mathrm{r}}(t)$:= number of motor units that are in a *resting state* at time t;
- $u(t)$:= muscular activation signal, sometimes referred to as the "brain stimulus" or "brain force";
- M := total number of motor units present in the muscles, assumed to be constant over time, i.e., $m_{\mathrm{a}}(t) + m_{\mathrm{f}}(t) + m_{\mathrm{r}}(t) = M$ for all t;

The adapted model is:

$$\dot{m}_{\mathrm{f}}(t) = -\theta_{\mathrm{f}\mapsto\mathrm{a}} m f(t) + \theta_{\mathrm{a}\mapsto\mathrm{f}} m_{\mathrm{a}}(t), \tag{1.1}$$

$$\dot{m}_{\mathrm{a}}(t) = -\theta_{\mathrm{a}\mapsto\mathrm{f}} m_{\mathrm{a}}(t) + \theta_{\mathrm{f}\mapsto\mathrm{a}} m_{\mathrm{f}}(t) + u(t)\theta_{\mathrm{r}\mapsto\mathrm{a}} m_{\mathrm{r}}(t) - (1-u(t))\theta_{\mathrm{a}\mapsto\mathrm{r}} m_{\mathrm{a}}(t), \tag{1.2}$$

$$m_{\mathrm{r}}(t) = M - m_{\mathrm{a}}(t) - m_{\mathrm{f}}(t), \tag{1.3}$$

Where the four parameters describe:

- $\theta_{\mathrm{f}\mapsto\mathrm{a}} > 0$ describes the recovery of fatigued motor units back into an active state;
- $\theta_{\mathrm{a}\mapsto\mathrm{f}} > 0$ captures the intuition that active motor units fatigue;
- $\theta_{\mathrm{a}\mapsto\mathrm{r}}$ describes how fast active muscles transition to resting state in case no brain force is applied;
- $\theta_{\mathrm{r}\mapsto\mathrm{a}}$ describes the recovery of resting motor units back into an active state.

The input term $u(t)m_{\mathrm{r}}(t)$ captures the fact that the brain stimulus $u(t)$ activates the resting motor units so that they become active.

To enable estimating the model from measured data using standard discrete-time approaches, it was performed a first order forward Euler discretization of the dynamics (1.1), (1.2) and (1.3), and obtain the piecewise-linear system:

$$\begin{cases} m_{\mathrm{f}}(k+1) = \phi_{\mathrm{f}\mapsto\mathrm{a}} m_{\mathrm{f}}(k) + (1-\phi_{\mathrm{a}\mapsto\mathrm{f}}) m_{\mathrm{a}}(k) \\ m_{\mathrm{a}}(k+1) = \phi_{\mathrm{a}\mapsto\mathrm{f}} m_{\mathrm{a}}(k) + (1-\phi_{\mathrm{f}\mapsto\mathrm{a}}) m_{\mathrm{f}}(k) + u(k)\phi_{\mathrm{r}\mapsto\mathrm{a}} m_{\mathrm{r}}(k) - (1-u(k))\phi_{\mathrm{a}\mapsto\mathrm{r}} m_{\mathrm{a}}(k) \\ m_{\mathrm{r}}(k) = M - m_{\mathrm{a}}(k) - m_{\mathrm{f}}(k) \end{cases} \tag{1.4}$$

where T is the length of the discretization period, and where the piecewise-linearity is induced by the fact that in our assumptions $u(k) \in \{0,1\}$, and the continuous and discrete-time parameters are connected by the relations:

$$\phi_{\mathrm{f}\mapsto\mathrm{a}} := 1 - \theta_{\mathrm{f}\mapsto\mathrm{a}} T, \qquad \phi_{\mathrm{a}\mapsto\mathrm{f}} := 1 - \theta_{\mathrm{a}\mapsto\mathrm{f}} T, \qquad \phi_{\mathrm{a}\mapsto\mathrm{r}} := \theta_{\mathrm{a}\mapsto\mathrm{r}} T, \qquad \phi_{\mathrm{r}\mapsto\mathrm{a}} := \theta_{\mathrm{r}\mapsto\mathrm{a}} T.$$

The results obtained after training the piecewise-linear model using experimental data shows that the model is capable of reproducing the main dynamics of fatiguing and recovering effects of the PFMs. In particular, the model is able to reproduce the the general

trend of the exerted muscular pressure and enables to predict the fatigue levels of the PFMs.

Note: Due to the purpose of the thesis which is the Bluetooth connection development, the section about the estimation problem and data preprocessing is not discussed here.

Chapter 2

Project description

In the first chapter we discussed the Kegeland application and the model of fatigue related to the PFMs. In this chapter the structure of the project and the technologies used to reach the objective set by the client will be discussed.
The client wants an application that works with both Android and on iOS devices.

The machine learning algorithms should be deployed to the cloud and the code should be retrieved from a version control system such as GitHub. If a researcher makes changes to the algorithms and publishes them to the cloud, the process of sharing these updates and using the new algorithms in the application should be automated. This also means that the user will automatically benefit from the new algorithms without updating their application.

The game should be customizable with different background images and level difficulties. In addition, a questionnaire will appear at the end of each game to monitor how the user is feeling. This data must be stored securely and the physicians need to be able to have access to this data. Physicians should also have access to more advanced statistics related to each patient. Users need to be able to see simple statistics on how they are improving.

2.1 Choices of Technical Solutions

To meet the goal of the project, the team has chosen the following technical solution such as libraries, frameworks and services.

2.1.1 Firebase

Firebase (Google 2021a) is a cloud service to build mobile and web applications. Firebase is created and maintained by Google, and it provides a wide range of services, such as cloud hosted databases, authentication through multiple providers, web storage and cloud functions. Firebase runs on Google Cloud which makes its services very reliable and accessible.

To store data, the team has chosen the document database called Cloud Firestore. Since Cloud Firestore is a non-relational document database, it provides dynamic and flexible data scheme's, which can comfortably handle a changing data model during the project life cycle.

Note: no costumer reference but put: the goal of the project was a recommender

The team used Firebase's built-in authentication with email and password as the main authentication provider, which ensures secure password hashing, and reliable web tokens for managing user sessions. The customer requested a recommender system that runs in a separate environment than the client application. Cloud Functions can trigger on changes in the database, get a request from an external service or execute certain operations on a specific time interval. The team chose to use Cloud Functions because it fulfilled all the requirements for the recommender system that the customer described to us in the beginning of the project.

One main advantage for using Firebase compared to other alternative Software as a service (SaaS) providers such as Microsoft Azure and Amazon Cloud, is that Firebase makes it very fast and easy to set up the cloud components. Firebase offers a lot more built-in functionality compared to Azure and Amazon, such as authentication, authorisation and web tokens for managing user sessions. Microsoft Azure and Amazon Cloud may provide more advanced features, but they also requires more effort to get up and running. Since the project had very simple cloud service requirements, Firebase was the best alternative because of all it's developer friendly features and effortless set-up.

2.1.2 React Native

React Native (Facebook 2021b) is a framework for building mobile applications. It is a hybrid framework, which makes it possible to run the same applications on both Android and iOS. The developers can write JavaScript/TypeScript, which will compile to native code at run-time. With a native framework for app development, the developers will usually have to write the code in both Java/Kotlin for Android and Swift/Objective-C for iOS. By using a hybrid framework such as React Native, the team got the benefit of

providing the application for both Android and iOS, without developing the application in two different developer environments.

React Native is provided by Facebook and it is very similar to React, which is a very popular library for creating web applications. The React Native developer community is immense, and there are numerous third-party libraries that can be used to achieve advanced functionality in very little time.

2.1.3 Jest

React Native ships with Jest (Facebook 2021a) as it's default testing framework, which makes it very easy to get started writing tests. Jest is a testing framework with focus on simplicity, easy configuration and a feature-rich API. Jest can be used for both unit- and state management testing.

2.1.4 Redux Toolkit

A very important part of an application is state management. The game which is already developed is based on Redux Toolkit (Abramov 2021) to handle state in the application because it is a robust framework with good documentation and a lot of developer resources. Redux Toolkit provides a global store, which will be provided to every component in the application. Every component can then subscribe to parts of the global state, which will trigger a re-render every time the state they are subscribing to changes, making it efficient and uncomplicated to create dynamic applications where user input or data changes are reflected throughout the application.

Redux Toolkit recommends to store data fetched from API requests, and its corresponding request status inside the global store. This makes it very easy to share data fetched from the database into the different components of the application. By storing the API request's status it is trivial to show loading spinners, disable buttons and show reassuring user feedback to enhance the usability of the application.

2.1.5 NativeBase

NativeBase is an accessible component library to build user interfaces with React Native and Web. NativeBase includes over 40 components such as buttons, forms and layout elements (NativeBase 2021). NativeBase assists developers to create a consistent user interface, consisting of ARIA accessible components.

2.1.6 React Native Game Engine

React Native Game Engine is a component based game library to construct "dynamic and interactive scenes using React Native" (Berak 2018). It runs a game engine, game loop and is responsible for controlling the interaction between game entities.

2.1.7 Matter

As physics engine, the team chose to use Matter.js (brm.io 2021) which works seamlessly together with React Native Game Engine. Matter offers eminent simulations of concepts found in the real world such as gravity, velocity and momentum. Programming a simulation of e.g., gravity to make it behave close to the real world is very challenging and time consuming, which makes Matter.js a very good choice to use as physics engine.

2.2 Use Case Diagram

Figure 2.1 illustrates the project's use case diagram. The project has two actors, User and Doctor. The user has access to the app through logging in, that allows them to start a game session or view statistics. The first time the user uses the application, they have to sign up. The doctor has access to a web dashboard where they can view their patients statistics and suggest exercises.

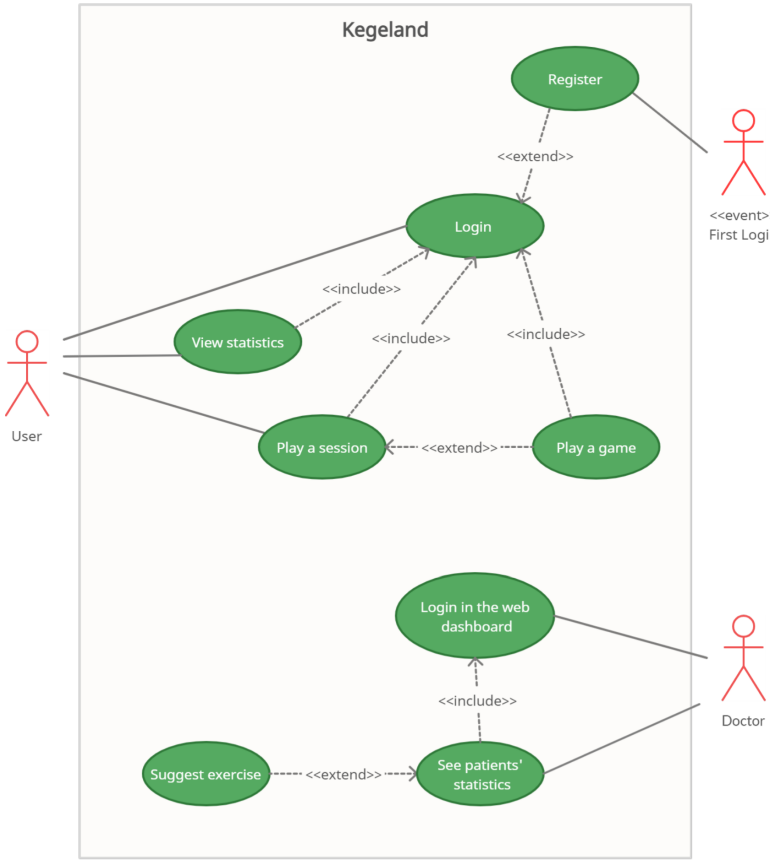

Figure 2.1: Use Case Diagram Source: [3]

2.3 Architectural Views

2.3.1 Process View

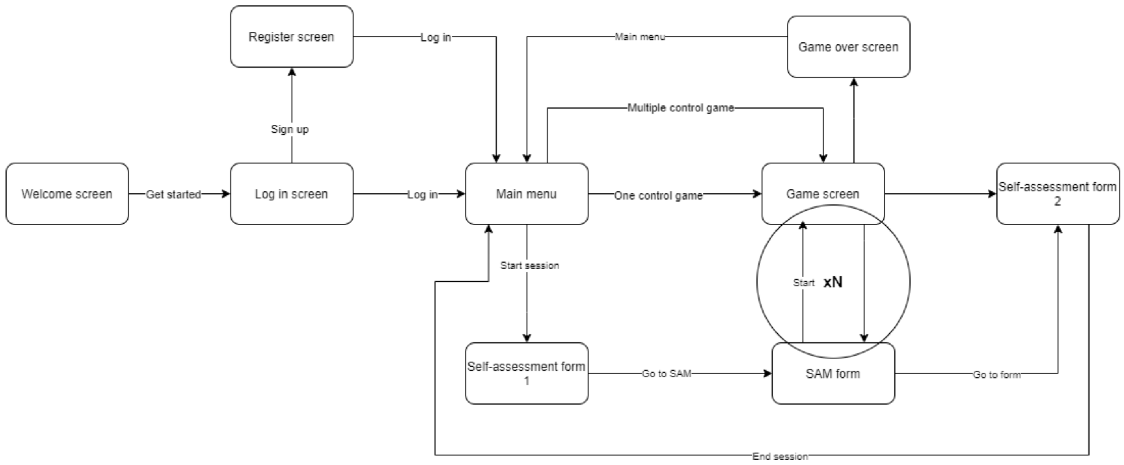

Figure 2.2: Process diagram. Source: [3]

This view shows how the user can transition between different states in the application. The user will start on the welcome screen, and by clicking on the "Get started"-button they are sent to the login screen. Here, the user can either log in with an existing user account, or click "Register" to go to the "Register"-screen. When logged in, the user will be at the main menu. Here they can either choose to play a single game, or start an exercise session. If the user plays a single game, they will be sent to the "Game Over"-screen after the game. In this screen, they can choose to go back to the main menu to start a new game. If they choose to play an exercise session, they will have to answer a self assessment form before and after the session. The exercise session consists of N games (where N is a modifiable parameter), where the user have to answer the SAM questionnaire before and after every game. After the last self assessment form, the user can go back to the main menu.

2.3.2 Physical view

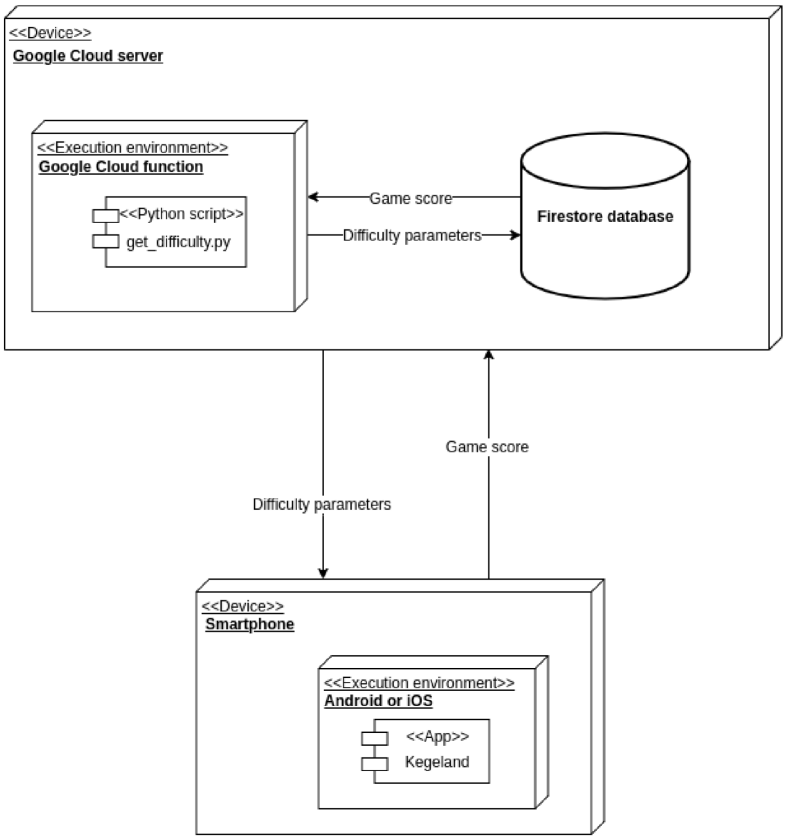

Figure 2.3: Physical view. Source: [3]

Figure 2.3 shows how our application is running on different physical devices. The user has the game installed on their Android or iOS device. The game communicates with a Google Cloud server which includes a Python script and a Firestore database. After every game the game score gets stored in the database. This triggers the Python script which will calculate the difficulty parameters for the user's next game. These parameters in turn are stored in the database. The next time the user starts a new game, the game application retrieves the updated difficulty parameters from the database so that they can be used in the game.

Chapter 3

Security

Security has not been a priority of the project is tested in a closed environment and is still in a very early phase. The team, however, advocated for implementing some security practices early on as it can be difficult to integrate security standards late in the project. Lack of security in an application can be critical and many companies experience the cost of a breach into their system. It is predicted that the cost of cyber crime in 2021 will reach 6 trillion USD (Morgan 2020). Being aware of these grim figures, the team chose to implement security through the use of third party providers.

Implementing a secure system can be a difficult and complex task. Getting it wrong can be critical, as it can leave the application vulnerable to attacks. Cloud providers such as Microsoft, Amazon and Google have solutions for security as they need to provide secure servers and data storage solutions to all their customers. By using Google Cloud services the team is secured through Google's security implementations, which provides much better and more reliable security than they would be able to implement ourselves.

3.1 Firebase Authentication

Firebase Authentication is a Google service providing authentication systems for many different uses (Google 2021a). They have many different ways of providing authentication, including Google Sign-in, but the team decided to go with e-mail and password as the customer requested it. It was also simple to set up and secure from the start, making it an optimal choice for the project.

3.2 Google Cloud Functions

Google Cloud Functions is a server-less solution for having code run in the cloud (Google 2021c). Google uses gVisor internally to ensure that functions cannot access other functions' operating systems (gVisor 2021). With this solution, Google handles the server for the team, meaning there is no need to worry about the security of the server. Considering they are a trusted provider, this is the most secure approach the group could have chosen as opposed to implementing a home-made security system for the project. This would most likely be less secure and more error-prone as the team lacks the expertise needed to implement a solid security system.

Chapter 4

Application screens

The following figures show the new version of the application, with screenshots of each section of the game: how to log in and register, main menu and the playing session.

4.1 Login and Registration

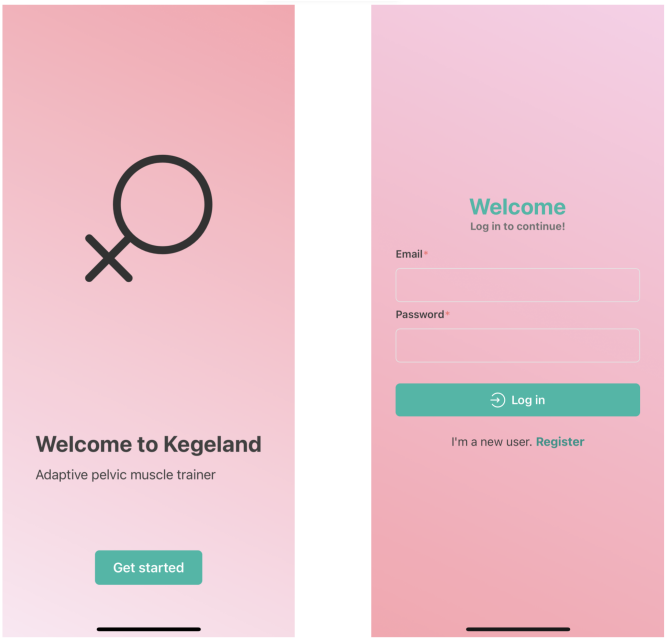

Figure 4.1: Home and Login Screens. Source: [3]

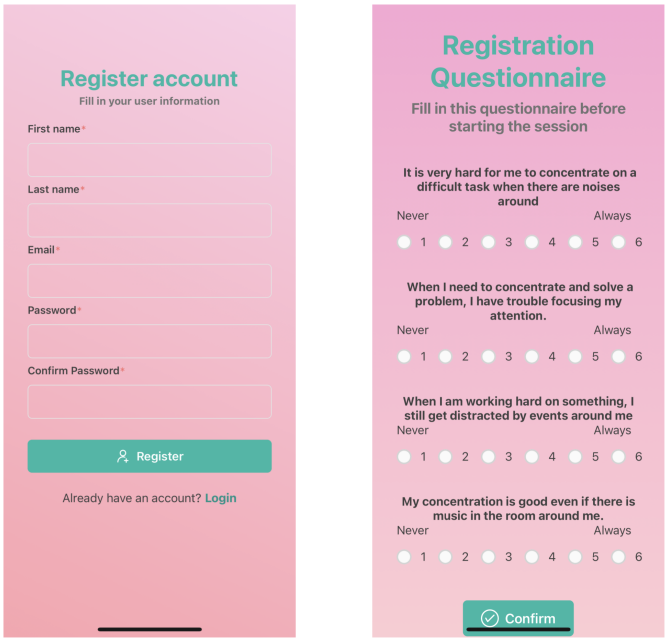

Figure 4.2: Registration Screens. Source: [3]

4.2 Main Menu

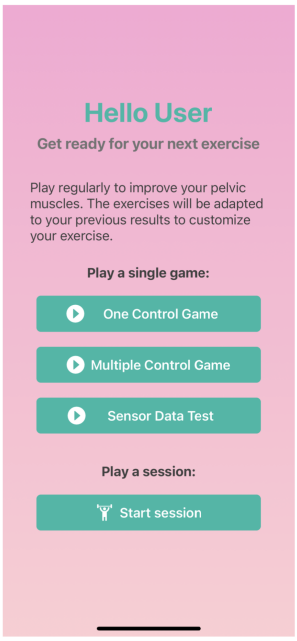

Figure 4.3: Main Menu Screen. Source: [3]

4.3 Game

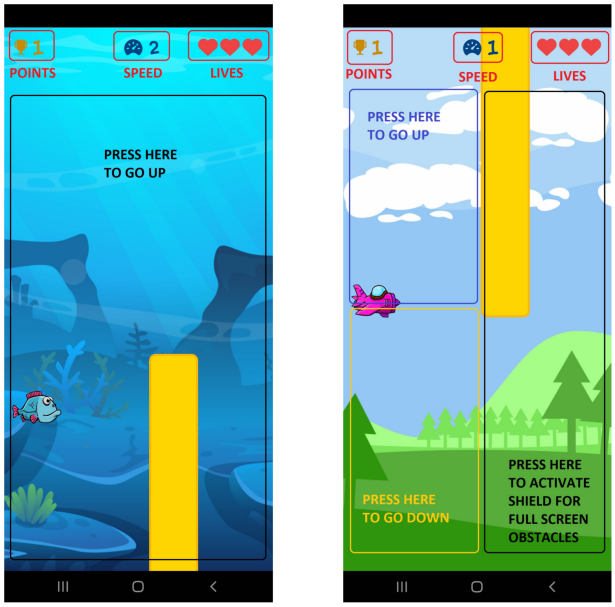

Figure 4.4: Game Screens. Source: [3]

Chapter 5

Bluetooth library development

The project goal was to deliver a game with a cloud backend server that is functional on both Android and iOS. The project met this goal, and a prototype was developed that can be used for further development in the research project.

Although the prototype is functional, there are still some functionalities and improvements that should be considered for the further development of the application. These are measures to ensure that the prototype can be used further in the research project.
One of which is the development of the Bluetooth library to connect the game with Femfit devices. Before starting the development I needed to understand the Bluetooth protocol, since the team had already developed an old version of the game that was based on Java, which is the language used for the development of the applications for Android smartphones.
In the old game the team had created a class from which they connected the game to the device, there I found that the Bluetooth protocol used was the new version of the protocol, called Bluetooth Low Energy (BLE).

5.1 BLE protocol

Bluetooth Low Energy (BLE), sometimes referred to as "Bluetooth Smart", is a lightweight subset of classic Bluetooth and was introduced as part of the Bluetooth 4.0 core specification [4].
The BLE standard was specifically designed for data exchange at low power and low cost for low-bandwidth applications.
The original target market for BLE technology was lower-rate data transfer for the personal smart device, smart home and fitness sensor markets [5].

5.1.1 GATT

GATT is an acronym for the Generic ATTribute Profile, and it defines the way that two Bluetooth Low Energy devices transfer data back and forth using concepts called Services and Characteristics. It makes use of a generic data protocol called the Attribute Protocol (ATT), which is used to store Services, Characteristics and related data in a simple lookup table using 16-bit IDs for each entry in the table (See figure 5.1) [6].

Figure 5.1: GATT, the Generic Attribute Profile, groups conceptually related attributes into a common parent container. Source: cardinalpeak.com

GATT services for Femfit device

The team used only two services from the Femfit device in the old application:

- *Pressure service, UUID: 0d9e0001-c111-49cd-bba3-85c7471cb6fa*: this service is used to measure the pressure of each sensor, with the following characteristics:
  1. *Pressure ch. 0-1, UUID: 0d9e0002-c111-49cd-bba3-85c7471cb6fa*: characteristic for the first sensor and the second one;
  2. *Pressure ch. 2-3, UUID: 0d9e0003-c111-49cd-bba3-85c7471cb6fa*: characteristic for the third sensor and the fourth one;
  3. *Pressure ch. 4-5, UUID: 0d9e0004-c111-49cd-bba3-85c7471cb6fa*: characteristic for the fifth sensor and the sixth one;
  4. *Pressure ch. 6-7, UUID: 0d9e0005-c111-49cd-bba3-85c7471cb6fa*: characteristic for the seventh sensor and the eighth one.
- *Housekeeping service, UUID: b92a0001-4bf9-4870-8aa1-881b3a20ada4*: this service is used to monitor the battery level of the device:
  - *Battery ch., UUID: b92a0002-4bf9-4870-8aa1-881b3a20ada4*: characteristic for the battery level.

5.2 Femfit BLE Test App

In order to test the library I have create a simple app with React Native, which is the same technology used by the team's new developed app.

5.2.1 Femfit Utils class

In the BLE Test App I have created a class called *FemfitUtils.ts* that contains the costants and functions used to connect the app to the Femfit device, for example:

- *MIN_PRESSURE*: the minimum pressure value that can be measured by the sensors;
- *MAX_PRESSURE*: the maximum pressure value that can be measured by the sensors;
- *HOUSEKEEPING_SERVICE*: the service UUID for the Housekeeping;
- *HOUSEKEEPING_CH*: array of characteristics UUID for the Housekeeping service;
- *PRESSURE_SERVICE*: the service UUID for the Pressure;
- *PRESSURE_CH*: array of characteristics UUID for the Pressure service;
- *convertPressure*: this function converts the pressure value from the Femfit device to a value that can be displayed in the app.
- *pressurePercentage*: this function converts the pressure value from the Femfit device to a value that can be displayed in the app.
- *convertSensorTemperature*: this function converts the temperature value from the Femfit device to a value that can be displayed in the app.
- *convertToBatteryVoltage*: this function converts the battery value from the Femfit device to a value that can be displayed in the app.
- *batteryPercentageForDisplay*: this function converts the battery value from the Femfit device to a value that can be displayed in the app.

```
import BatteryStatus from "./../types/BatteryStatus.enum";

const HOUSEKEEPING_SERVICE = "b92a0001-4bf9-4870-8aa1-881b3a20ada4"; //battery charge level
const HOUSEKEEPING_CH = "b92a0002-4bf9-4870-8aa1-881b3a20ada4";

const PRESSURE_SERVICE = "0d9e0001-c111-49cd-bba3-85c7471cb6fa";
const PRESSURE_CH = [
  "0d9e0002-c111-49cd-bba3-85c7471cb6fa", //1 and 2
  "0d9e0003-c111-49cd-bba3-85c7471cb6fa", //3 and 4
  "0d9e0004-c111-49cd-bba3-85c7471cb6fa", //5 and 6
  "0d9e0005-c111-49cd-bba3-85c7471cb6fa", //7 and 8
  "0d9e0006-c111-49cd-bba3-85c7471cb6fa",
];

const MIN_PRESSURE = 760;
const MAX_PRESSURE = 970;

const convertPressure = (b1: number, b2: number) => {
  return parseFloat(
   (--------).toPrecision(8)
  );
};

const pressurePercentage = (pressure: number) => {
  if (pressure < MIN_PRESSURE) return 0;
  else if (pressure > MAX_PRESSURE) return 100;
  else
    return parseInt(
      ((pressure - MIN_PRESSURE) / (MAX_PRESSURE - MIN_PRESSURE)).toFixed(0)
    );
};

const convertSensorTemperature = (b: number) => {
  return -------;
};

const convertToBatteryVoltage = (b: number, b2: number) => {
  return (--------);

const batteryPercentageForDisplay = (d: number) => {

  if (d <= 3.5) {
    return BatteryStatus.VeryLow;
  }
  if (d <= 3.65) {
    return BatteryStatus.Low;
  }
  if (d <= 3.75) {
    return BatteryStatus.Medium;
  }
  if (d <= 3.85) {
    return BatteryStatus.MediumHigh;
  }
  if (d <= 3.95) {
    return BatteryStatus.High;
  }
  if (d <= 4.1) {
    return BatteryStatus.Full;
  }
  return BatteryStatus.Unknown;
};

export {
  MIN_PRESSURE,
  MAX_PRESSURE,
  HOUSEKEEPING_SERVICE,
  HOUSEKEEPING_CH,
  PRESSURE_SERVICE,
  PRESSURE_CH,
  convertPressure,
  pressurePercentage,
  convertSensorTemperature,
  convertToBatteryVoltage,
  batteryPercentageForDisplay,
};
```

5.2.2 React Context API

Context provides a way to pass data through the component tree without having to pass props down manually at every level.
In a typical React application, data is passed top-down (parent to child) via props, also known as prop drilling, but such usage can be cumbersome for certain types of props (e.g. locale preference, UI theme) that are required by many components within an application. Context provides a way to share values like these between components without having to explicitly pass a prop through every level of the tree [7].

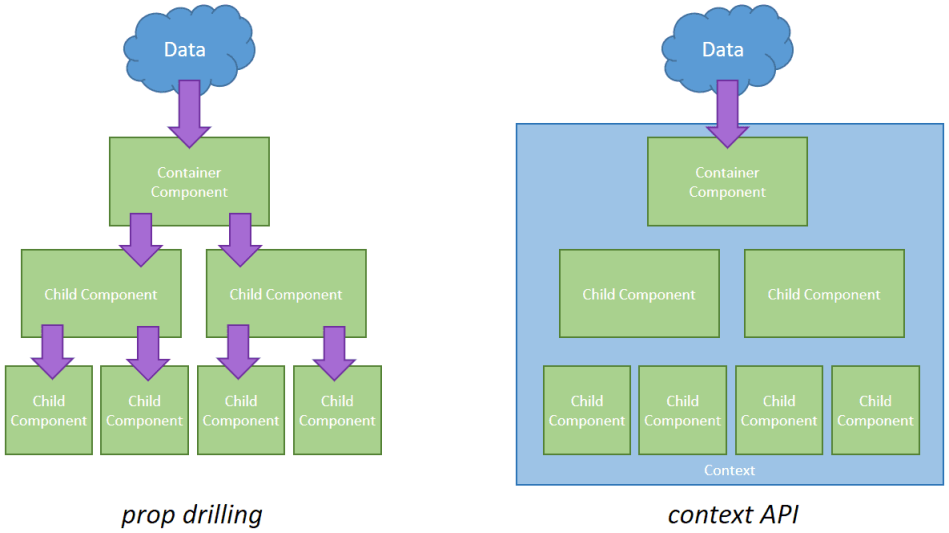

Figure 5.2: Props drilling vs Context API - Source: `reactjs.org`

FemfitProvider

The *FemfitProvider* is the class that implements the Context API. Here I have used an already existing library for managing the BLE connection, which is called *react-native-ble-manager* [8].
In the FemfitProvider I have made available the following functions:

- *batteryStatus*: this function returns the battery status of the Femfit device.
- *bluetoothStatus*: this function returns the bluetooth status of the Femfit device.
- *isSearchingDevice*: this function returns a boolean value that indicates if the app is searching for the Femfit device.

- *peripheral*: this function returns the peripheral of the device, the peripheral contains the information about the device id, name, received signal strength indicator (RSSI), and also the services and characteristics of the device.
- *searchFemfit*: this function starts the search for the Femfit device.
- *startMonitoring*: this function starts the monitoring of the Femfit's Pressure and Battery level characteristics.
- *stopMonitoring*: this function stops the monitoring of the Femfit's Pressure and Battery level characteristics.
- *isMonitoring*: this function returns a boolean value that indicates if the app is monitoring the Femfit device status.
- *disconnectFemfit*: this function disconnects the Femfit device.
- *getPressuresAndTemperatures*: this function returns the pressure and temperature values of all the sensors as an array.

```tsx
import React, { useState, useContext, createContext, useEffect } from "react";
import {
  NativeEventEmitter,
  NativeModules,
  PermissionsAndroid,
  Platform,
} from "react-native";
import BleManager from "react-native-ble-manager";
import BluetoothStatus from "../types/BluetoothStatus.enum";
import FemfitContext from "../types/FemfitContext";
import Peripheral from "../types/Peripheral.interface";
import BatteryStatus from "../types/BatteryStatus.enum";
import {
  batteryPercentageForDisplay,
  convertPressure,
  convertSensorTemperature,
  convertToBatteryVoltage,
  HOUSEKEEPING_CH,
  HOUSEKEEPING_SERVICE,
  pressurePercentage,
  PRESSURE_CH,
  PRESSURE_SERVICE,
} from "../utils/FemfitUtil";

const femfitContext = createContext({} as FemfitContext);

export function FemfitProvider(props: any) {
  const ffP = useProvideFemfit();

  return (
    <femfitContext.Provider value={ffP}>
      {props.children}
    </femfitContext.Provider>
  );
}

export const useFemfit = () => {
  return useContext(femfitContext);
};

const BleManagerModule = NativeModules.BleManager;
const bleEmitter = new NativeEventEmitter(BleManagerModule);

function useProvideFemfit() {
  const [isSearchingDevice, setIsSearchingDevice] = useState(false);
  const [isMonitoring, setIsMonitoring] = useState(false);
  const [bluetoothStatus, setBluetoothStatus] = useState(BluetoothStatus.Idle);
  const [peripheral, setPeripheral] = useState<Peripheral>({} as Peripheral);
  const [batteryStatus, setBatteryStatus] = useState(BatteryStatus.Unknown);
  const [pressureS1, setPressureS1] = useState(0);
  const [pressureS2, setPressureS2] = useState(0);
  const [pressureS3, setPressureS3] = useState(0);
  const [pressureS4, setPressureS4] = useState(0);
  const [pressureS5, setPressureS5] = useState(0);
  const [pressureS6, setPressureS6] = useState(0);
  const [pressureS7, setPressureS7] = useState(0);
  const [pressureS8, setPressureS8] = useState(0);
  const [temperatureS1, setTemperatureS1] = useState(0);
  const [temperatureS2, setTemperatureS2] = useState(0);
  const [temperatureS3, setTemperatureS3] = useState(0);
  const [temperatureS4, setTemperatureS4] = useState(0);
  const [temperatureS5, setTemperatureS5] = useState(0);
  const [temperatureS6, setTemperatureS6] = useState(0);
  const [temperatureS7, setTemperatureS7] = useState(0);
  const [temperatureS8, setTemperatureS8] = useState(0);

  useEffect(() => {
    console.log("Mount");

    // initialize BLE modules
    BleManager.start({ showAlert: false });

    // add ble listeners on mount
    bleEmitter.addListener(
      "BleManagerDiscoverPeripheral",
      handleDiscoverPeripheral
```

```
    );
    bleEmitter.addListener("BleManagerStopScan", handleStopScan);
    bleEmitter.addListener(
      "BleManagerDisconnectPeripheral",
      handleDisconnectedPeripheral
    );
    bleEmitter.addListener(
      "BleManagerDidUpdateValueForCharacteristic",
      handleUpdateValueForCharacteristic
    );

    // check location permission only for android device
    if (Platform.OS === "android" && Platform.Version >= 23) {
      PermissionsAndroid.check(
        PermissionsAndroid.PERMISSIONS.ACCESS_FINE_LOCATION
      ).then((r1) => {
        if (r1) {
          console.log("Permission is OK");
          return;
        }

        PermissionsAndroid.request(
          PermissionsAndroid.PERMISSIONS.ACCESS_FINE_LOCATION
        ).then((r2) => {
          if (r2) {
            console.log("User accept");
            return;
          }

          console.log("User refuse");
        });
      });
    }

    // remove ble listeners on unmount
    return () => {
      console.log("Unmount");

      bleEmitter.removeAllListeners("BleManagerDiscoverPeripheral");
      bleEmitter.removeAllListeners("BleManagerStopScan");
      bleEmitter.removeAllListeners("BleManagerDisconnectPeripheral");
      bleEmitter.removeAllListeners(
        "BleManagerDidUpdateValueForCharacteristic"
      );
    };
  }, []);

  const handleDiscoverPeripheral = async (peripheral: Peripheral) => {
    if (!peripheral.name) {
      peripheral.name = "NO NAME";
      console.log("TROVATO", peripheral.name);
    }

    setPeripheral(peripheral);

    await BleManager.connect(peripheral.id);

    BleManager.isPeripheralConnected(peripheral.id, []).then((isConnected) => {
      if (isConnected) {
        setBluetoothStatus(BluetoothStatus.Connected);

        console.log("Peripheral is connected!");
      } else {
        setBluetoothStatus(BluetoothStatus.Disconnected);
        console.log("Peripheral is NOT connected!");
      }
    });
  };

  const handleStopScan = () => {
    console.log("Scan is stopped");
    setIsSearchingDevice(false);
  };

  // handle disconnected peripheral
  const handleDisconnectedPeripheral = (data: any) => {
    console.log("Disconnected from peripheral: ", data.peripheral);
```

```
154
155     setBluetoothStatus(BluetoothStatus.Disconnected);
156     setPeripheral({} as Peripheral);
157   };
158
159   const handleUpdateValueForCharacteristic = ({
160 ,     value,
161 ,     peripheral,
162 ,     characteristic,
163 ,     service,
164 ,   }: {
165 ,     value: any;
166 ,     peripheral: string;
167 ,     characteristic: string;
168 ,     service: string;
169 ,   }) => {
170     try {
171       if (
172         service === HOUSEKEEPING_SERVICE &&
173         characteristic === HOUSEKEEPING_CH
174       ) {
175         let cBv = convertToBatteryVoltage(value[4], value[5]);
176         let bPd = batteryPercentageForDisplay(cBv);
177         // let temp= convertTemperature(value[6], value[7]);
178
179         setBatteryStatus(bPd);
180       } else {
181         // let counter = value[1] & 255;
182
183         let channel = PRESSURE_CH.findIndex((ch) => ch === characteristic);
184
185         if (channel === -1) return;
186
187         let sensor1 = channel * 2;
188
189         //pressures samples
190         let sample1s1 = convertPressure(value[2], value[3]);
191         let sample2s1 = convertPressure(value[5], value[6]);
192         let sample3s1 = convertPressure(value[8], value[9]);
193
194         let sample1s2 = convertPressure(value[11], value[12]);
195         let sample2s2 = convertPressure(value[14], value[15]);
196         let sample3s2 = convertPressure(value[17], value[18]);
197
198         let samplesTemperature1s1 = convertSensorTemperature(value[4]);
199         let samplesTemperature2s1 = convertSensorTemperature(value[7]);
200         let samplesTemperature3s1 = convertSensorTemperature(value[10]);
201
202         let samplesTemperature1s2 = convertSensorTemperature(value[13]);
203         let samplesTemperature2s2 = convertSensorTemperature(value[16]);
204         let samplesTemperature3s2 = convertSensorTemperature(value[19]);
205
206         let averageA = parseInt(
207           ((sample1s1 + sample2s1 + sample3s1) / 3).toFixed(0)
208         );
209         let averageB = parseInt(
210           ((sample1s2 + sample2s2 + sample3s2) / 3).toFixed(0)
211         );
212
213         let averageC = parseInt(
214           (
215             (samplesTemperature1s1 +
216               samplesTemperature2s1 +
217               samplesTemperature3s1) /
218             3
219           ).toFixed(0)
220         );
221         let averageD = parseInt(
222           (
223             (samplesTemperature1s2 +
224               samplesTemperature2s2 +
225               samplesTemperature3s2) /
226             3
227           ).toFixed(0)
228         );
229
```

```
230         if (sensor1 === 0) {
231           setPressureS1(pressurePercentage(averageA));
232           setPressureS2(pressurePercentage(averageB));
233           setTemperatureS1(averageC);
234           setTemperatureS2(averageD);
235         } else if (sensor1 === 2) {
236           setPressureS3(pressurePercentage(averageA));
237           setPressureS4(pressurePercentage(averageB));
238           setTemperatureS3(averageC);
239           setTemperatureS4(averageD);
240         } else if (sensor1 === 4) {
241           setPressureS5(pressurePercentage(averageA));
242           setPressureS6(pressurePercentage(averageB));
243           setTemperatureS5(averageC);
244           setTemperatureS6(averageD);
245         } else if (sensor1 === 6) {
246           setPressureS7(pressurePercentage(averageA));
247           setPressureS8(pressurePercentage(averageB));
248           setTemperatureS7(averageC);
249           setTemperatureS8(averageD);
250         }
251       }
252     } catch (e) {
253       console.log(e);
254     }
255   };
256
257   const searchFemfit = () => {
258     // skip if scan process is currenly happening
259     if (isSearchingDevice) {
260       return;
261     }
262
263     setPeripheral({} as Peripheral);
264
265     // then re-scan it
266     BleManager.scan([HOUSEKEEPING_SERVICE, PRESSURE_SERVICE], 5, true)
267       .then(() => {
268         console.log("Scanning...");
269         setIsSearchingDevice(true);
270       })
271       .catch((err: any) => {
272         console.error(err);
273       });
274   };
275
276   const startMonitoring = async () => {
277     if (!peripheral.id) return;
278
279     await BleManager.retrieveServices(peripheral.id, [
280       PRESSURE_SERVICE,
281       HOUSEKEEPING_SERVICE,
282     ]);
283
284     PRESSURE_CH.forEach((ch) => {
285       BleManager.startNotification(peripheral.id, PRESSURE_SERVICE, ch).then(
286         () => {
287           console.log("Started notification on " + ch);
288         }
289       );
290     });
291
292     BleManager.startNotification(
293       peripheral.id,
294       HOUSEKEEPING_SERVICE,
295       HOUSEKEEPING_CH
296     ).then(() => {
297       console.log("Started notification on " + HOUSEKEEPING_CH);
298     });
299
300     setIsMonitoring(true);
301   };
302
303   const stopMonitoring = async () => {
304     if (!peripheral.id) return;
305
306     PRESSURE_CH.forEach((ch) => {
```

```
307       BleManager.stopNotification(peripheral.id, PRESSURE_SERVICE, ch).then(
308         () => {
309           console.log("Stopped notification on " + ch);
310         }
311       );
312     });
313
314     BleManager.stopNotification(
315       peripheral.id,
316       HOUSEKEEPING_SERVICE,
317       HOUSEKEEPING_CH
318     ).then(() => {
319       console.log("Stopped notification on " + HOUSEKEEPING_CH);
320     });
321
322     setIsMonitoring(false);
323   };
324
325   const disconnectFemfit = () => {
326     if (!peripheral.id) return;
327
328     BleManager.disconnect(peripheral.id).then(() => {
329       console.log("Disconnected from " + peripheral.id);
330       setBluetoothStatus(BluetoothStatus.Disconnected);
331       setPeripheral({} as Peripheral);
332     });
333   };
334
335   const getPressuresAndTemperatures = () => {
336     let pt = [
337       [pressureS1, temperatureS1],
338       [pressureS2, temperatureS2],
339       [pressureS3, temperatureS3],
340       [pressureS4, temperatureS4],
341       [pressureS5, temperatureS5],
342       [pressureS6, temperatureS6],
343       [pressureS7, temperatureS7],
344       [pressureS8, temperatureS8],
345     ];
346
347     return pt;
348   };
349
350   return {
351     batteryStatus,
352     bluetoothStatus,
353     isSearchingDevice,
354     peripheral,
355     searchFemfit,
356     startMonitoring,
357     stopMonitoring,
358     isMonitoring,
359     disconnectFemfit,
360     getPressuresAndTemperatures,
361   };
362 }
```

5.2.3 Application test

Finally I have created two simple views that are used to test the library:

Femfit connection (Main screen)

In the *Main screen* we found a button in the middle of the screen that starts the search for the Femfit device.

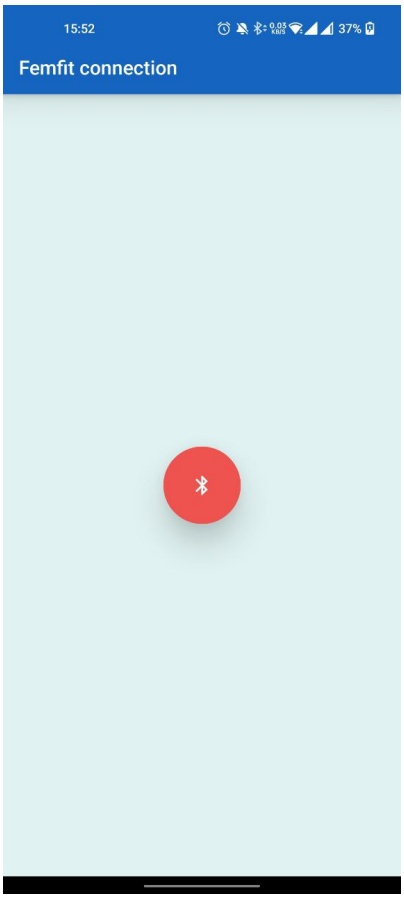

Figure 5.3: Main screen

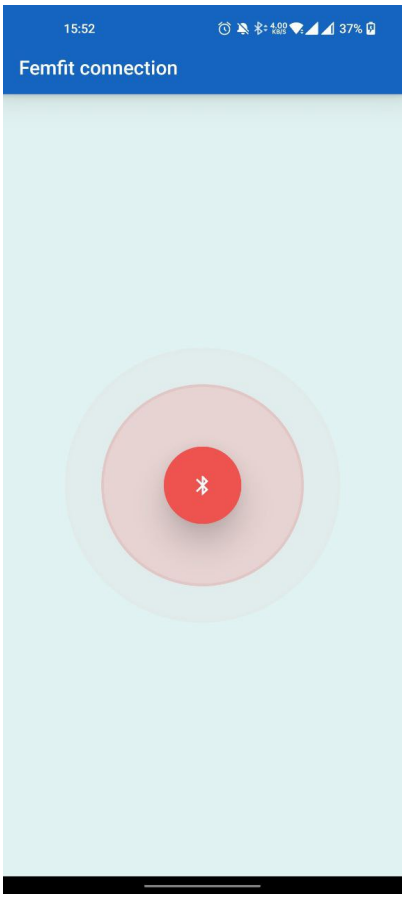

Figure 5.4: Searching started

After pressing the button a ripple effect is shown around the button, which indicates that the search for the Femfit device has started (see figure 5.4).

Pressures status

When the search for the Femfit device is finished, and the device is found, the app shows the device's information, such as name and the MAC address. We will not see the battery

status as well as the pressure and temperature values immediately, because the monitoring of the device is not started yet. In the top right corner of the screen we found a button that disconnects the device from the app.

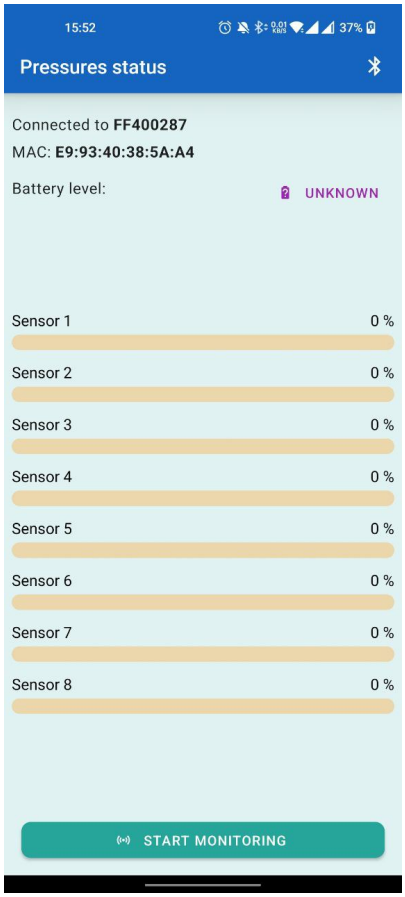

Figure 5.5: Pressures details

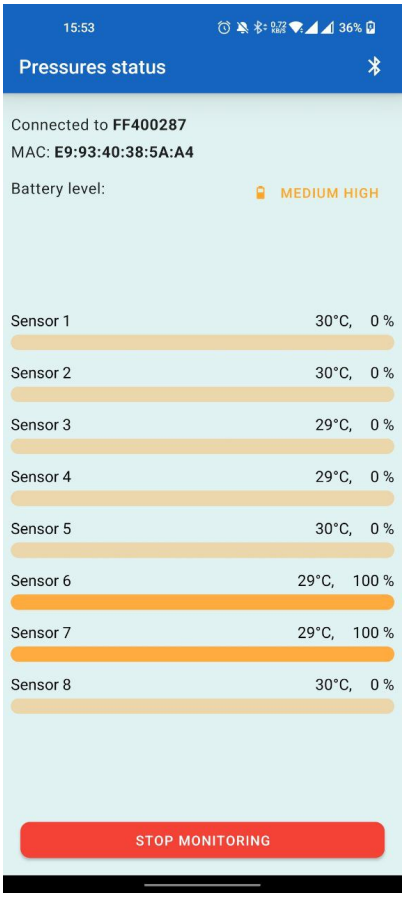

Figure 5.6: Monitoring started

At the bottom of the screen we found a button that starts the monitoring of the device (see figure 5.5).

Chapter 6

Conclusion

In this thesis we developed a new version of the game app for controlling the PFM. The importance of Kegel exercises and why they need to be done, and why they need to be done under the supervision of a doctor has been discussed. How studying the fatigue modeling problem of PFM can help assess players' health. It was discussed why the team decided to create a new App, and the technologies used and the architecture of the app.

Finally I have discussed the Bluetooth technology and the development of the BLE library for the Femfit device.
The Bluetooth library functionalities are working as requested, also the information as well as the data given by the sensors are shown in the BLE Test App. The project is not completed yet, since the library needs to be imported into the Kegeland App and tested with actual players (e.g. patients). Until now the game was tested with mockup data recorded from the Femfit device. I have made a documentation of the library and a video where I explain how the library works, this will help the next person who's going to be part of the project to understand and implement the library into the Kegeland App.

The BLE Test App was shown as a proof of concept, it's not a finished product, but it's a good example of how the library can be used, the code was also given to the client to help them to understand how the library works.

References

[1] N. Kas, D. M. Budgett, J. Kruger, P. M. F. Nielsen, D. Varagnolo, S. Knorn, *Data-driven modelling of fatigue in pelvic floor muscles when performing Kegel exercises*, 2019.

[2] J. Liu, R. Brown, and G. Yue, *A dynamical model of muscle activation, fatigue, and recovery*, Biophysical Journal, 2002.

[3] I. Crivellari, S. Dahl, S. Husnes, T. Ravndal, T. Woldseth, *Practical Assignment Adaptive Pelvic Floor Muscles Trainer*, 2021.

[4] *Introduction to bluetooth low energy*, `https://learn.adafruit.com/introduction-to-bluetooth-low-energy/introduction`

[5] *What is ble, and how do its related gap and gatt profiles work?* `https://www.cardinalpeak.com/blog/what-is-ble-and-how-do-its-related-gap-and-gatt-profiles-work`

[6] *GATT*, `https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gatt`

[7] *Context*, `https://reactjs.org/docs/context.html`

[8] *React Native BLE manager*, `https://github.com/innoveit/react-native-ble-manager`