NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE

JOÃO PEDRO DOS SANTOS PERES

Bachelor in Computer Science and Engineering

KNEEMOR

AN AUTONOMOUS PHYSIOTHERAPY AND MONITORING SYSTEM FOR TRACKING KNEE MOBILITY

MASTER IN COMPUTER SCIENCE

NOVA University Lisbon
September, 2024

NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE

KNEEMOR

AN AUTONOMOUS PHYSIOTHERAPY AND MONITORING SYSTEM FOR TRACKING KNEE MOBILITY

JOÃO PEDRO DOS SANTOS PERES

Bachelor in Computer Science and Engineering

Adviser: Carmen Pires Morgado
Assistant Professor, NOVA School of Science and Technology

Co-adviser: Fernanda Barbosa
Assistant Professor, NOVA School of Science and Technology

Examination Committee

Chair: Artur Miguel de Andrade Vieira Dias
Assistant Professor, NOVA School of Science and Technology

Members: Carlos Jorge de Sousa Gonçalves
Assistant Professor, Instituto Superior de Engenharia de Lisboa

Carmen Pires Morgado
Assistant Professor, NOVA School of Science and Technology

MASTER IN COMPUTER SCIENCE

NOVA University Lisbon
September, 2024

KNEEMOR

An autonomous physiotherapy and monitoring system for tracking knee mobility

This document was created with the (pdf/Xe/Lua)LaTeX processor and the NOVAthesis template (v6.10.7) [43].

Acknowledgements

It has been a long and difficult journey on reaching this point and achieving this goal whilst working full-time and also managing other aspects and challenges in my life.

The conclusion of this chapter in my life would've not been possible without some honorable mentions of the people that made it possible and that paved the way for my success.

I would like to start by thanking my advisor Carmen Morgado and my co-adviser Fernanda Barbosa, for displaying formidable patience and compassion with me, guiding me in the best way possible whenever they could and most importantly, never giving up on me despite my late deliveries and overall insecurities.

I can say I'm humbled and proud for having studied Integrated Master's in Computer Science in the NOVA School of Science and Technology, due to its brilliant teachers that instilled in me the tools and knowledge necessary for thriving in the software engineering field, and also for providing life long advice's that I intend to follow and keep to my heart. Also during this academic venture, I had the opportunity of crossing paths and collaborating with incredible colleagues, where we struggled, laughed, and overcame obstacles together. Some of which as of today, I may call them friends, thank you for supporting me and providing me with your friendship.

I would also like to thank all the test subjects that provided a foundation for the evaluation section, and also the four physiotherapists that kindly participated in the physiotherapist questionnaires and also for being available to give insights and also feedback on the developed system.

I'd like to also extend my gratitude to my employer, OMIP, for providing me with the time and flexibility to pursue this lifelong dream while working, and also for allowing me to meet incredible colleagues that motivated and supported me throughout this journey even in the days that I felt like giving up, thank you for everything and believing in me above all.

Last but not least, I would like to thank my family, closest friends and also my cats, for providing me unconditional support and being there in the hardest of times, without you I wouldn't be where I am.

"He, who has a why to live for, can bear with almost any how. "
(Friedrich Nietzsche)

Abstract

As our world heals from the pandemic crisis that *COVID-19* wrought in these past years, more and more business models and services are slowly being reinvented and re-adapted to remote solutions that still provide value but at the distance of the end-user's fingertips, like mobile applications [42]. Although many business models can be easily ported to a remote solution approach, other types of services like medical care tend to pose physical challenges, namely physiotherapy.

A physiotherapist typically has to be present in order to interact with the patients joints and limbs so it can accurately diagnose the patients current pathology, prescribe the proper protocol for his rehabilitation, monitor his evolution and if needed re-calibrate certain exercises based on the patients current condition.

With these challenges in mind, comes the implementation of a telerehabilitation system named *KNEEMOR* which is an autonomous physiotherapy and monitoring system for tracking knee flexion/extension range of motion (ROM).

Autonomous, because patients are able to do their rehabilitation exercises without the presence of a physiotherapist thanks to feedback provided by a mobile application and also through a prototype embedded with actuators and sensors that are used for capturing the knee flexion/extension motion.

Monitoring, since physiotherapists can keep track of patient's knee mobility progress, provide partial diagnosis at a distance, prescribe the proper exercises for the patient's condition, re-calibrate these exercises if needed and have a historical view of the overall progression of the patient's knee ROM. A patient can monitor his knee ROM progress, view the prescribed protocols and exercises, request medical appointments, export both exercise and medical data in various formats, and also visualize the captured angle from the prototype whilst performing an rehabilitation exercise, indicating if he performed the movement correctly and if he has finished the exercise, hence promoting the concept of patient autonomy.

Keywords: Physiotherapy, Remote Physiotherapy, Knee Rehabilitation, Telerehabilitation, Telehealth, MHealth, Motion Sensors

Resumo

Com o nosso mundo a recuperar da crise pandêmica que o *COVID-19* provocou nestes últimos anos, modelos de negócio e serviços começam pouco a pouco a serem reinventados e adaptados a soluções remotas que continuem a oferecer o mesmo valor mas que estejam à distância de um click dos utilizadores, como é o caso das aplicações móveis [42].

Apesar de muitos modelos de negócio serem facilmente adaptáveis para uma solução remota, existem serviços como a medicina onde não é tão linear fazer esta transição, especialmente em especialidades que envolvam intervenção física como é o caso da fisioterapia.

Um fisioterapeuta normalmente tem que interagir com os membros e articulações dos pacientes, de forma a conseguir diagnosticar corretamente e por sua vez prescrever os exercícios mais apropriados para a reabilitação do paciente, reabilitação que tipicamente ocorre em clínica sob a presença do fisioterapeuta.

Com estes desafios em mente, implementou-se um sistema de tele-reabilitação chamado *KNEEMOR*, que consiste num sistema autónomo de fisioterapia e de monitorização sobre a mobilidade do joelho.

Autónomo no aspeto de que, os pacientes podem realizar exercícios de reabilitação sem a presença de um fisioterapeuta, mas sim com o apoio de uma aplicação móvel e de um protótipo sensorial que dá feedback auditivo e visual quando o paciente realiza um exercício.

Monitorização, visto que os fisioterapeutas podem efetuar a gestão dos vários pacientes remotamente, visualizar e observar métricas recolhidas do protótipo usado pelos pacientes, efetuar diagnósticos à distância, atribuir exercícios a pacientes e reajustar esses exercícios conforme a evolução do paciente e extrair relatórios referentes ao progresso do paciente. O paciente pode também monitorizar o seu progresso, visualizar os seus exercícios atribuídos, requisitar consultas médicas e extrair também relatórios referentes a exercícios efetuados.

Palavras-chave: Fisioterapia, Fisioterapia Remota, Reabilitação do joelho, Tele-reabilitação, Telemedicina, Medicina Móvel, Sensores Inerciais,

Contents

List of Figures

[1] Rotation Matrix - image from: https://en.wikipedia.org/wiki/Rotation_matrix

List of Tables

Acronyms

AAROM active-assistive range of motion *(p. 12)*
ACL anterior cruciate ligament *(p. 11)*
ADC analog-to-digital converter *(pp. 18, 48)*
AHRS attitude and heading reference system *(p. 20)*
API Application Programming Interface *(p. 71)*
APK Android Application Package *(p. 107)*
AROM active range of motion *(p. 12)*
ATT Attribute Protocol *(pp. 55, 56)*

BLE bluetooth low energy *(p. 64)*

CORS cross-origin resource sharing *(p. 95)*
CQRS command query responsability segregation *(p. 82)*
CQS command query separation *(p. 82)*
CSS Cascading Style Sheets *(p. 31)*

DAC digital-to-analog converter *(p. 48)*
DAO data access objects *(pp. 40, 81, 83)*
DI dependency injection *(p. 82)*
DMP digital motion processor *(pp. 23, 52, 79)*
DOF degrees of freedom *(pp. 10, 20, 51)*
DOM Document Object Model *(p. 33)*
DTO data transfer objects *(pp. 40, 82, 83)*

EMF electromagnetic fields *(p. 19)*

GAP Generic Access Protocol *(pp. 55, 56)*
GATT Generic Attribute Protocol *(pp. 55, 56)*

HMAC-SHA256 Hash-based Message Authentication Code using the Secure Hashing Algorithm 256 bit hashing function *(p. 93)*

HMAC-SHA512 Hash-based Message Authentication Code using the Secure Hashing Algorithm 512 bit hashing function *(p. 93)*

HTTP Hypertext Transfer Protocol *(p. 90)*

i18n internationalization and localization *(p. 97)*

IMU inertial measurement unit *(pp. 6, 24)*

IoHT Internet of Health Things *(p. 3)*

IoMT Internet of Medical Things *(p. 3)*

IoT Internet of Things *(p. 2)*

JSX Javascript XML *(p. 33)*

JWT JSON Web Token *(pp. 82, 83)*

L2CAP Logical Link Control and Adaptation Protocol *(p. 55)*

LCL lateral collateral ligament *(p. 11)*

LED light emmiting diode *(p. 52)*

MCL medial collateral ligament *(p. 11)*

MEMS micro-electromechanical system *(p. 20)*

mHealth Mobile Health *(p. 3)*

MOCAP motion capture *(p. 16)*

MTU Maximum Transmission Unit *(p. 79)*

PBKDF2 Password-Based Key Derivation Function 2 *(p. 93)*

PCL posterior cruciate ligament *(p. 11)*

PROM passive range of motion *(p. 12)*

PWA progressive web app *(pp. 32, 62, 102)*

REST Representational State Transfer *(p. 71)*

RMSE root mean square error *(p. 115)*

ROM range of motion *(p. 12)*

RPC remote procedure calls *(p. 47)*

SMP Security Manager Protocol *(pp. 55, 56)*

SPA single page application *(p. 62)*

SUS system usability scale *(p. 118)*

UI user interface *(pp. 5, 64)*

UUID Universally Unique Identifier *(p. 56)*

VDOM Virtual Document Object Model *(p. 33)*

1

Introduction

In this introductory chapter, the motivation for a tele-rehabilitation system such as *KNEEMOR* is explored in depth, describing the current challenges that both patients and medical practitioners face when providing remote consultations, the potential benefits and disadvantages a system like this brings to the table, and the overall importance for the creation of the system.

Moving on into the context section, a brief rundown on the keywords meaning is provided, to better contextualize the reader and to improve the comprehension of the document on the main topics that will be discussed.

Following this, the goals defined to achieve with this thesis are outlined, including the expected results and the planned deliverables. Finally, the structure of the thesis document is briefly discussed in order to provide guidance and also to facilitate the navigation of the reader throughout the document.

1.1 Motivation

Every day, as technology advances, we are driven toward a more automated way of life, removing effort and physical interaction in tasks that previously required our intervention. Services are slowly becoming accessible at our fingertips due to the introduction of mobile devices and web applications in our lives. From food services, shopping services, booking services, and even medical services are starting to require less and less physical presence from the users, which eases the overall physical labor for tasks in our daily life but as a consequence increases the overall sedentarism.

In the case of medical services, developments for remote health technologies have increased dramatically with the latest pandemic incident, which forced doctors and patients alike to adopt new strategies in order to mitigate the contagion of *COVID-19* to a minimum, while also maintaining and providing the needed medical care that patients required [57].

Also in some specialities like the case of **physiotherapy**, where this practice requires a more physical intervention with the patients, video conferencing consultations might not

always be the most viable solution for diagnosing and making the required treatments in the long run. Additionally since most of the traditional assessments are made in medical facilities and in-person by a physiotherapist, only discrete time framed assessments are performed, which can be detrimental if the prescribed exercises are degrading patients health instead of improving it. The geographical factor of attending consultations is also another concern, where not every patient has the best mobility options in terms of body motion, or have the economic means to keep visiting medical facilities just to attend consultations in person. Traditional physiotherapy doesn't offer proper ways to monitor and assist patients continuously outside of medical environments [3].

This causes the need for tele-rehabilitation systems that provide continuous monitoring and feedback to patients when undergoing their recovery programs. With the reduced costs of mobile devices, Internet of Things (IoT) systems components also becoming cheaper and cloud solutions becoming ever more available to the public, these ingredients pave the way for development of robust tele-rehabilitation monitoring systems that address the limitations previously discussed.

1.2 Context

In this section, a brief introduction to the keywords from the abstract is provided whilst also giving clarity on the differences between terms. Afterwards, a discussion is made over the role of tele-rehabilitation in IoT systems and also their use in mobile applications.

1.2.1 Keywords

As technology developments are propelled to new heights of maturity as years go by, new approaches for services begin to arise, deprecating older architectures and systems with better solutions. Physical services start to shift to a remote way of operation and in the case of the medical systems, we start to see a larger adherence, specially due to the recent events from the *COVID-19* pandemic which caused a lot of presential constraints for attending patients health, all in the efforts of decreasing the risk of rising the overall contagion [57].

When researching for remote health systems, this topic often comes across with other terms like *telemedicine, mHealth, telehealth* and more recently *IoMT and IoHT*. These terms while being strongly correlated to medical sciences and care, mean different things.

Starting with the broader term which is **telehealth**, encompasses any technology that aims to provide some sort of care remotely, let it be via telecommunications, services or even electronic devices. Telehealth isn't constrained only to medical practices but can entail other types of areas like fitness training, elderly monitoring, animal care, health promoting, etc [33].

Telemedicine on the other hand, is a subset from telehealth which only comprises the area of medical services that provide care at-a-distance with similar approaches as

telehealth. Telemedicine applications have a broad scope from handling patients health track data in databases and services, video conference enabled consultations, online diagnosis tools and even remote treatment devices that can be adjusted or controlled by medical staff.

Also with the continuous rising adoption and accessibility to smartphones, the term **Mobile Health (mHealth)** begins to arise [78]. The concept behind mHealth is the same as telehealth but with the exception that applications are exclusively deployed to mobile devices and also the care provided by these applications typically falls entirely on the responsibility of the user, meaning mHealth targets more self-care. Nowadays, mHealth is becoming a standard in our lives since in most smartphone operating systems [46], have preinstalled health monitoring applications by default, and additionally a broad library of fitness and health tracking applications on their application stores.

Also with this trend, mHealth applications started being incorporated in even smaller devices than typical smartphones like the case of smartwatches. These deliver more limited functionalities compared to smartphones, but as technology advances more and more features start becoming available, which means that the scale of devices is becoming gradually smaller while providing the same functionalities as previous generations of devices. With the previous mention of smaller dimension mobile devices, smaller ways of providing medical monitoring and care start to emerge thanks to IoT technology and also cloud services exposure, creating the new term **Internet of Medical Things (IoMT)** or also known as **Internet of Health Things (IoHT)**.

Due to the existence of these new technological approaches, physical rehabilitation starts migrating into **tele-rehabilitation** or also known as **e-rehabilitation** [66]. This form of rehabilitation is delivered through digital means, rather than the traditional in-person intervention and guidance provided by physiotherapists.

While tele-rehabilitation offers a more comfortable and convenient way for patients to perform their rehabilitation, and also by removing the geographical constraints and travel costs to medical facilities, it still allows for clinical assessment and therapy through motion monitoring and feedback mechanisms that provide guidance to patients during their therapy. This approach is also beneficial to medical practitioners [13], since it allows them to provide rehabilitation sessions remotely, giving them more flexibility and freedom of how to do their practice, and also since these systems rely on patient health tracking, the medical practitioners can also assess the patient's progress through digital means instead of relying on a physical examination at all times.

1.2.2 Tele-rehabilitation in IoHT and mHealth

Health care services along the years have improved, mainly due to the technological advances and specially through digitalization of the health information.

Ranging from electronic medical records, electronic prescriptions, remote appointment booking, and even disease prevention systems, all these functionalities improve overall

quality and efficiency in health care [51]. Similarly, tele-rehabilitation systems improve medical care by offering significant benefits to patients, particularly patients who live far from medical facilities or that have limited mobility, as they no longer need to travel for their rehabilitation sessions, requiring only an initial consultation to assess their initial health conditions.

However, in order to have a tele-rehabilitation system that allows patients to do their prescribed protocol from their home environments, there are various challenges that must be overcome. For autonomous rehabilitation, a patient must be able to perform the exercises in a proper manner without assistance of the physiotherapist but through some sort of feedback mechanism (visual, auditive or even through vibration). Also another issue with tele-rehabilitation is the fact that remaining faithful to the prescribed exercise program after an accident or surgery to limb, tends to be hard to promote outside a medical environment as well as remembering the right schedules to perform the exercises. This brings the need of smart remote physiotherapy systems that can provide solutions to these problems [3].

Postolache G. et al. identified the several requirements in order to have a reliable IoT-based information system [51]. These findings laid foundation for a information system that can be used for tele-rehabilitation.

One of the requirements was the need for multi-modality of input device reception meaning that a IoT information system should be able to receive data from multiple input sources (peripherals and motion capture) [51]. Also it should be easy to incorporate new types of input devices, using a unique identifier system to discriminate both old and new devices while remaining platform agnostic and supporting older standards. The same concern would be applied to IoT communication, an IoT information system should provide a wide range of communication protocols while also assessing their impact on the overall quality of service of the IoT information system. As for data transmission from one machine to another machine in a IoT system, the techniques that should be employed are through the use of event publishing or polling. Additionally, the data storage and processing should be promoted to cloud services, typically a backend server, to perform the needed data processing and store it safely in database, removing the overhead on IoT sensor networks [51]. The database should store both data collected from the IoT sensors, as well as personal data regarding subjects that interact with the IoT system. Through the means of the frontend layer, it should be possible to parameterize the various metrics to affect the IoT sensors and it should also drive user engagement through the use of serious games techniques like real-time 3D representation of exercises or through video-based guidance.

Another big concern in terms of IoT rehabilitation systems is security [51]. Security and privacy should be a concern from the start of the development since having poor security in a medical system can be disastrous and originate in both patient and personnel data leakage. When developing any system, security should be taken in account in every layer, however implementing security also tends to cause sometimes disruption of functionalities

and also affect the performance on systems, and since IoT resources are more limited, this requirement must be tailored carefully to ensure that these systems continue to function correctly without heavy processing overheads.

Despite the various benefits that remote physiotherapy brings, the adoption of mHealth solutions for physiotherapists is still lacking [42]. Most physiotherapists don't rely on tele-rehabilitation systems due to several factors: the poor accuracy and precision of the data collected from sensors, the lack of integration of the data collected into medical record systems, the data regulations involving data extraction, storage and sharing, the overall work needed to validate such applications for professional use and also the price tag on acquiring and setting up a system like this. Physiotherapists however tend to encourage *mHealth* applications mostly for education purposes and exclude the potential of mobile applications use for overall rehabilitation.

1.3 Thesis Goals

As seen previously, remote health systems encompass a multitude of various terms and use different types of architectures, having each their own use cases and inherent challenges. Most of these systems gradually start becoming less intrusive and obtrusive in patient's daily lives, through the combined use of IoT technologies, cloud services and smartphone usage that is ever more available these days. The combination of these technologies provide monitoring, assessment, pathology prevention and rehabilitation which are crucial key factors for general health care. Consequently, these systems reduce operational costs of having specialized medical equipment that is normally only present in medical environments, and also removes the geographical condition which affects both patients and health practitioners when having to attend clinical visits, again reducing costs for both parties.

With these challenges in mind comes this thesis, whose main objective is to develop an autonomous remote physiotherapy and monitoring system for monitoring the extension/flexion range of motion on the knee joint through the performance of three knee exercises: heel slides, standing knee flexion and sitting knee flexion.

The autonomy part of this system should provide functionality to guide patients in their exercise programs, correct them while they perform such exercises via audio and visual feedback and also should promote them to follow their protocols in the right time schedules, whilst making them engaged in the overall rehabilitation process through a effective and interactive user interface (UI). The interactive UI will provide visual feedback to patients, promote their overall engagement in the rehabilitation process through reminders and exercise schedules. The physiotherapy and monitoring system should also provide ease of communication between patients and physiotherapists, provide the needed knee angle data to assess patient's conditions and progress, allow the scheduling of appointments, keep track of the historical data from the patients progress and allow physiotherapists to intervene remotely on rehabilitation programs if needed by changing

the parameterizations of the assigned exercises (like maximum and minimum angles for range of motion) and also assigning different types of exercises. The knee angle data collection from patients will be through the use of inertial measurement unit (IMU) sensors, which will allow physiotherapists to assess the patient's current condition on continuous terms instead of discrete time frames. Through this motion tracking, it provides better decision making and guidance on the types of exercises the patients should do and also enables physiotherapists to validate if patients are performing the exercises according to the right medical instructions. The motion sensors will provide a angle that is formed between the thigh and the shank, that is the knee flexion/extension angle which is the angle we are interested in measuring. Despite the knee having six degrees of freedom of motion (three rotations and three translations), the main degree of freedom that will be studied and applied in this system will be the rotation that is parallel to the sagittal plane, which is knee flexion and extension. Along with the motion sensors, two frontend applications (mobile application for patients and progressive web application for physiotherapists) and a backend application will be developed to sustain this system.

1.4 Document Structure

The document structure is divided in six main chapters, with the first chapter providing the motivation, importance and the overall context of this thesis.

The second chapter of the document encompasses the academic research based on physiotherapy concepts, remote physiotherapy systems and the state of the art solutions (both academic and commercial) to perform *tele-rehabilitation* and to track knee joint motion. At the end of the second chapter, conclusions are made from the researched topics.

The third chapter describes the thesis proposal, which contains the description of the core functionalities the system should provide, the overall system architecture and its underlying components.

The fourth chapter materializes the proposed functionalities and architecture into an system implementation. Here it is discussed the implementation details of all components in-depth, the problems that arose during development and the decisions made when implementing the various features.

The fifth chapter explains how the system is validated and tested as a whole. Also we discuss the various collected results from performing the validation on the system and also insights from questionnaires.

On the sixth and final chapter of the document it is discussed the conclusions made with the development of this work, and also it is explored the improvements and newer features that could be adopted in future work.

2

State of the Art

This chapter entails some introductory concepts around knee physiotherapy, including knee anatomy, knee rehabilitation exercises based on specific knee pathologies and measurement angles regarding knee mobility. In this chapter are also introduced the various reference systems, explaining how they work and their relationship to sensor motion capturing. Additionally, the chapter also covers the technical state of the art in measuring knee flexion and extension angles, existing systems for rehabilitation and measurement of joint angles, and the types of hardware components and feedback mechanisms that can be used during exercises.

Also, health care system design guidelines are explored, along with the development technologies and frameworks best suited for building the *KNEEMOR* system. The possible architectures and design patterns to be applied in the backend, as well as the necessary communication protocols are also discussed. At the end of the chapter, an analysis is performed over some existing commercial solutions for providing rehabilitation and conclusions are drawn from the state of the art research.

2.1 Introductory Anatomy

In this section some introductory concepts about anatomy and physiotherapy are introduced to the reader, so that there is a better comprehension of the physiotherapy area, the knee anatomy and what angles are measured to assess knee range of motion.

2.1.1 Anatomy Concepts

For anatomic body references [8], typically three imaginary planes are used to refer structures of the human body as depicted in Figure 2.1: the sagittal/median plane which is the vertical plane that intersects the body dividing it into a left and right section, the coronal/frontal plane (also a vertical plane) which divides the body into a front section and back section and lastly the transverse/axial plane which is the perpendicular plane to both previous planes and which splits the human body into upper section and lower section.

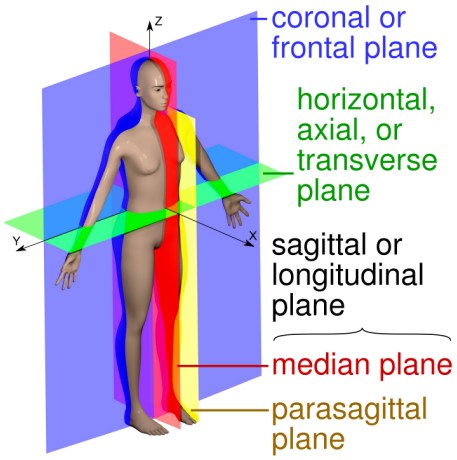


Figure 2.1: Anatomic planes [8]

It's also a practice in anatomy to describe the location of body parts in relation to a imaginary line that is drawn through the middle of the human body which can be correlated to the sagittal plane. For instance, the term medial means closer to the midline, so when referring to the medial side of the knee, we refer to the side that is closest to other opposite knee [50]. In contrast the lateral side is the side which is furthest away from the other knee. Another two terms that are commonly used in medicine literature are distal and proximal, where distal refers to "distance" or far from a particular spot, and proximal stands for proximity hence meaning some part of the body that is closer to another.

Also in anatomical terms, the front side of the knee is referred to as the anterior, while the backside is referred to as the posterior. The human body has various joints, some of them are particular special and essential for overall motion since they allow bones to glide on each other, these joints are named as synovial joints. A synovial joint is typically composed of two or more bones and provides motion between them without originating friction [37].

The friction is mitigated due to the joints composition as seen in Figure 2.2:

- Hyaline cartilage (or Articular Cartilage), a special membrane that surrounds the end surfaces that are connected to other bones and that provides friction reduction and shock absorption.
- Joint capsule, an fibrous compartment filled with synovial fluid that encapsulates a joint. It keeps the bones in place with the aid of ligaments, tendons and muscles.
- Synovial fluid, a sticky substance that provides nutrients and lubrication for cartilage surfaces, hence allowing bones to glide and move on each other.
- Synovium, a special membrane of cells that produces the synovial fluid.

A hinge joint, which is also a synovial joint is a joint which provides most of its motion only in one plane and allows small degrees of motion in others [26].

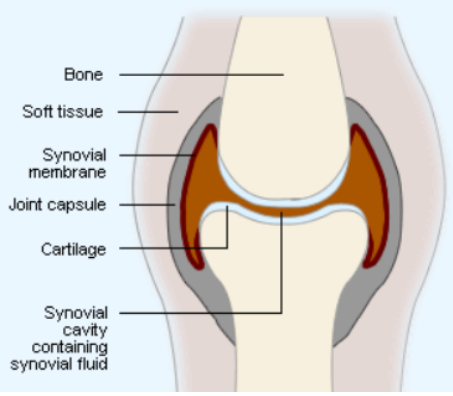


Figure 2.2: Synovial joint structure [37]

2.1.2 Knee Anatomy

In order to propose a system that helps in the knee health monitoring and rehabilitation, we must first dive into the anatomy of the knee and study its overall structures.

The knee joint is considered one of the most important joints in the human body since without it our movement becomes impaired or even completely disabled. It's one of the hinge joints from the human body and it provides motion mostly in one degree of freedom, i.e., one plane or one dimension, through **knee flexion and extension**. In total, the knee has six degrees of freedom (DOF), composed by three rotations (flexion and extension, varus/valgus abduction and adduction, external and internal rotation) and three translations (anterior/posterior glide, medial/lateral shift, compression/distraction) as depicted in Figure 2.3.

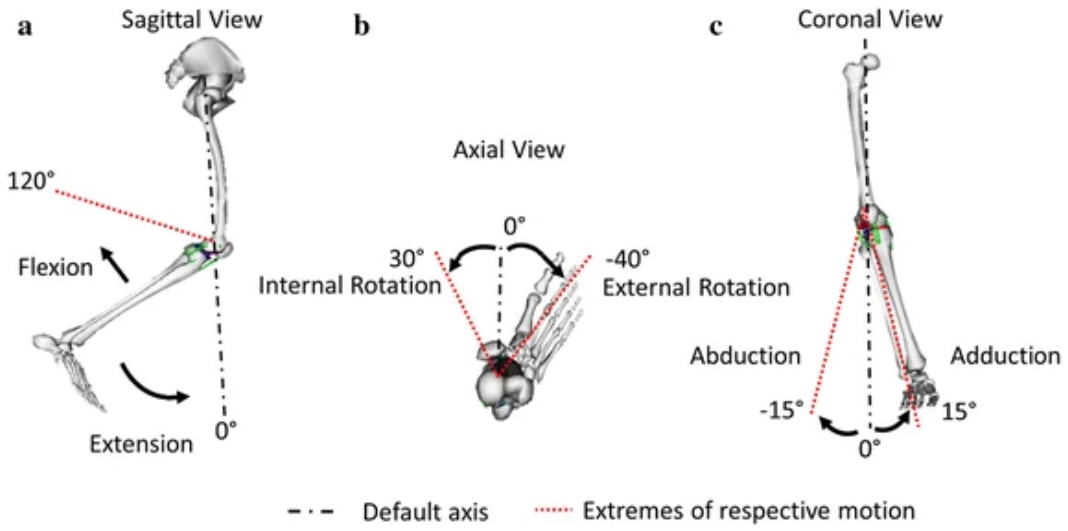


Figure 2.3: Main knee joint rotations [44]

The knee joint can withstand forces up to 4-6 times our own body weight, hence it must adhere to a robust composition to hold the joint intact and enable its overall function

[67].

The knee, apart from the joint, is also composed of bones, ligaments and tendons, muscles, nerves, blood vessels.

The knee joint is created via connection of the bottom of the femur (or thigh bone) to the top surface of the tibia (or shin bone) as displayed in Figure 2.4. The two round knobs located in the femur's bottom are called femoral condyles which rest on the tibia's surface. The patella which sits in front of the knee, glides through a groove that is formed by these femoral condyles. The fibula which is the smaller bone near the tibia doesn't belong to the knee joint [50].

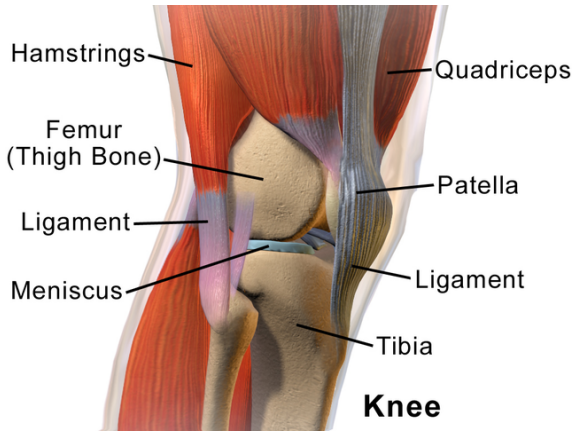


Figure 2.4: Overview of knee joint structure [67]

Articular cartilages cover the meeting surfaces of the femur and tibia to allow them to glide on each other, and also provide shock absorption. On the patella the cartilage is present behind it. On the joint capsule resides synovial fluid which supplies nutrients and lubrication to the surrounding cartilages.

The ligaments are bands of tissue that help keeping the bones together in place and to provide knee joint stability [29].

There are four main ligaments as seen in Figure 2.5: medial collateral ligament (MCL), lateral collateral ligament (LCL), anterior cruciate ligament (ACL) and posterior cruciate ligament (PCL).

The MCL and LCL ligaments, which are located in the sides of the knee and connected to the femur and tibia, prevent the knee of bending too much inwards and outwards. The ACL and PCL ligaments which run between the femoral condyles, prevent the tibia of moving too far forward and backwards in relation to the femur.

Additionally there are another two special fibrous cartilages which are called the menisci.

The lateral meniscus and medial meniscus are crucial for the knee joint since they help spread the human body weight around the joint. If the menisci didn't exist, the whole weight would be concentrated to one point on the tibia creating too much stress on the bone and potentially damaging it.

The menisci enables the knee to turn a few degrees inward and outward through the shallow socket it forms on the tibia. This socket is what enables the femur to have motion and rotation on certain degrees without having friction on the tibia.

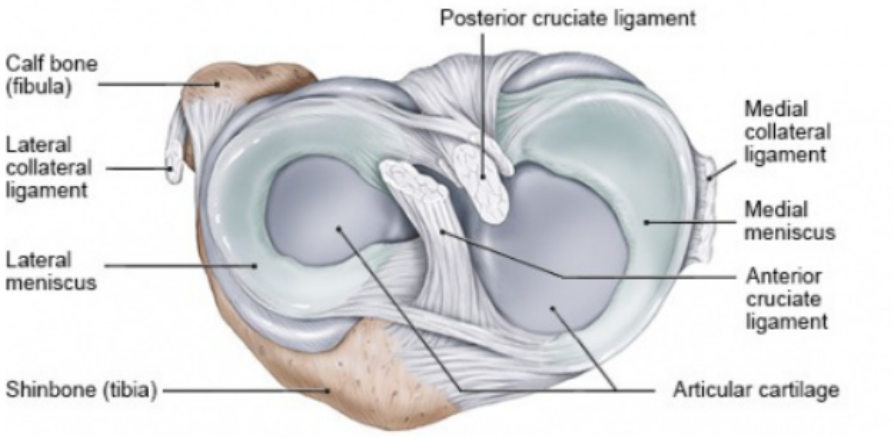


Figure 2.5: Cross section view of a knee joint from above [37]

On the knee there is also tendons which are fibrous tissues that attach bones to muscles, with the two main tendons being the patellar tendon and the quadriceps tendon. Both are attached to the patella but the patellar tendon is also attached to the tibia [29].

The muscles surrounding the knee are the quadriceps muscles and hamstring muscles. When the former muscle contracts it causes the knee to straighten, while when the latter contracts it causes bending. This forms the extensor mechanism which drives the knee joint and provides us with the ability to walk.

The popliteal nerve, which provides sensation and muscle control, travels along the lower leg and foot and splits above the knee to form the tibial nerve and the peroneal nerve.

The blood supply of the knee is delivered through the popliteal arteries and popliteal veins which are vital for leg and consequently knee functioning. Without a popliteal artery functioning correctly it can lead to leg decay which then results in amputation.

2.1.2.1 Knee range of motion and angles of measurement

The term range of motion (ROM) consists of maximum length that a joint or muscle can move in various directions.

In physiotherapy, active range of motion (AROM) is defined as the maximum length a joint can move on its own without external assistance, while passive range of motion (PROM) refers to the maximum length a joint can move relying on external assistance let it be manual assistance or mechanical and active-assistive range of motion (AAROM) to the maximum length a joint can move with some degree of assistance to prevent further injury.

The measurement of range of motion is typically accomplished by the use of a goniometer which is an handheld device with two rulers that measures angles [80]. A physiotherapist can measure these three types of ROM in different contexts. The first context is for the PROM measurement which is made with a patient that has his injured joint fully relaxed and the motion is created manually by a physiotherapist or through a machine. In the case of AAROM, the measurement is made when a patient can move his injured joint partially but requires external help to carry on the motion lest the injury is aggravated. Lastly the AROM measurement is made when the patient uses only his muscles to move the injured joint without any assistance from the physiotherapist or a machine.

When addressing knee range of motion, there are three main motions that are assessed, knee flexion which consists of the action of bending the knee, knee extension which consists in straightening the knee and also a slight knee rotation. A healthy knee with full extension refers to 0 degrees while a full flexion of the knee refers up to 140 degrees. As for knee rotation, the rotation inwards goes up to 30 degrees, while outwards reaches 40 degrees [44].

2.1.2.2 Types of exercises for knee rehabilitation/strengthening

A physiotherapist on a patient's first appointment normally asks about the history of the problem, the aggravating and relieving factors and past medical history which might be correlated to the current pathology. Afterwards, a series of tests are conducted to perform a full examination on the knee. The main examinations are the ones that follow:

- gait evaluation, an assessment of how a patient walks since a poor form on walking can contribute to knee strain and stress;
- palpation examination, an assessment which involves the use of hands to touch the many structures entailed to the knee, in order to detect knee abnormalities and pinpoint pain areas;
- range of motion measurements, which assess the maximum length a knee can extend or flex;
- strength measurements, to detect muscular weakness or imbalance that might provoke knee pain;
- balance assessment, since imbalance when walking or having a tendency to be supported more on only one side of the body can cause excessive strain and stress to the knee causing pain;
- girth or swelling measurements, which aim to check for inflammations or potential internal hemorrhaging.

Despite alternative methods being available like ultrasound, electric stimulation, P.R.I.C.E and R.I.C.E. (which stands for protection, rest, ice, compression and elevation), kinesiology tapping and even soft tissue massaging and joint mobilization, it has been proven that having a well tailored training program prescribed by a physiotherapist yields great results in overall rehabilitation and is also crucial in certain situations, for example, for surgeries where the muscles tend to lose strength [80].

Rehabilitation exercises can be broken down into two categories: strength exercises and stretch exercises. For a proper rehabilitation of the knee, an optimal exercise program should address typically: hip, knee and ankle range of motion and also include gradual strengthening exercises. The exercises should always be performed in a pain free manner, slowly and with good form in order to achieve better healing results. When an exercise starts being a prejudice, it should always be readjusted so that the patient's knee's health doesn't degrade.

Strength exercises for the knee typically target the quadriceps and hamstrings, but in some situations having a good core, hip and glute strength can provide control and balance to the knee [40]. Stretching exercises are aimed for building flexibility which as a consequence provide an increase of range of motion to the knee.

Figure 2.6: Knee exercises and the corresponding angles to be measured: heel slides, standing knee flexion and sitting knee extension

Due to the nature of the thesis and since we will be monitoring mostly range of motion exercises, Saba Bakhshi et. al. found that the optimal home exercises for a patient who had knee surgery should be the standing knee bending, standing hip bending, sitting

knee extension and heel slides [38].

From the previous exercises, in the developed work we will only focus on monitoring the range of motion angles on three exercises: standing knee bending, sitting knee extension and heel slides as illustrated in Figure 2.6.

2.2 Motion Capture Fundamentals and Reference Systems

Before addressing existing knee angle measurement systems, it is imperative to have a basic understanding of motion capture, the existing techniques to accomplish it and brief breakdown on the advantages and disadvantages of each technique to formalize the selected choice for the developed work.

Correlated to motion capture, it is also important to know how orientation and rotation is defined on a rigid body. In this section, besides going through motion capture techniques, the different mathematical reference representations that are commonly used to describe orientation and rotation on rigid bodies will also be explored, which will be fundamental for the developed work of the thesis.

2.2.1 Motion Capture and Motion Tracking Techniques

After the brief introduction to knee anatomy, there's now a basic understanding of how the knee joint works and which key anatomical references are needed to track the knee movement (specifically the thigh and shin). In order to track the knee joint angle, the basics of motion capture must be explored, since these bring forth a baseline of how the knee angle measurement will be accomplished.

The term motion capture (MOCAP) refers to the process of capturing real world physical movement and translating it into digital means. For capturing physical movement, MOCAP systems rely on motion tracking techniques that contain one or more sensors that are able to perceive the physical movement that is happening in the real world.

These motion tracking techniques have been classified over the years by various researchers and most techniques can be classified into two main groups: **optical motion tracking systems and non-optical motion tracking systems**[56].

2.2.1.1 Optical Motion Tracking Systems

Optical motion tracking systems are systems which perceive physical movement through video recording from a set of cameras. In contrast to photogrammetery which uses photographs from multiple angles to measure and create three dimensional objects, an optical motion tracking system uses video cameras to measure and track rigid bodies either through **a marker-based approach or a markerless-based approach** [79].

The gold standard approach for measuring three dimensional kinematics which is **marker-based**, consists on tracking the position and orientation of rigid bodies through the use of light emitting markers or reflective markers [39]. In the case of reflective markers (or passive markers), the cameras emit infrared light which is then reflected by the markers that have retro-reflective material. The latter case which is light emitting markers do not require the infrared lights since these markers already emit light themselves.

By displacing multiple cameras around multiple angles and through the brightness that markers bring to the environment, it is possible to triangulate the marker's position

and orientation while motion is being performed.

The biggest advantage of this approach is that it provides very accurate results for position and orientation of objects and as such provides the best three dimensional kinematics measuring. As for disadvantages, these types of systems are typically very expensive, have very specific room requirements in order to build the motion tracking system and they are time consuming since during motion capture sessions it requires previous preparation before starting the motion tracking (setting up the camera angles, marker placement, lighting adjustments).

The markerless-based approach instead relies on the power of computer vision algorithms and related techniques to analyze movement and patterns based on frames from video recordings.

An example of a markerless-based solution was the work from Scott D. Uhlrich et al. ***OpenCap*** [72], a open-source framework for motion capture which employed computer vision algorithms to determine both kinematics and dynamics of human movement using videos captured from two or more smartphones. Pose estimation algorithms were used to identify body limbs from videos, deep learning and biomechanical models to estimate the three-dimensional kinematics and lastly physics-based simulations to provide estimation of the muscle activations and musculoskeletal dynamics. In this work a comparative cost analysis between a traditional marker motion capture solution and ***OpenCap*** is presented which enforces what was previously stated on the disadvantages of marker-based systems.

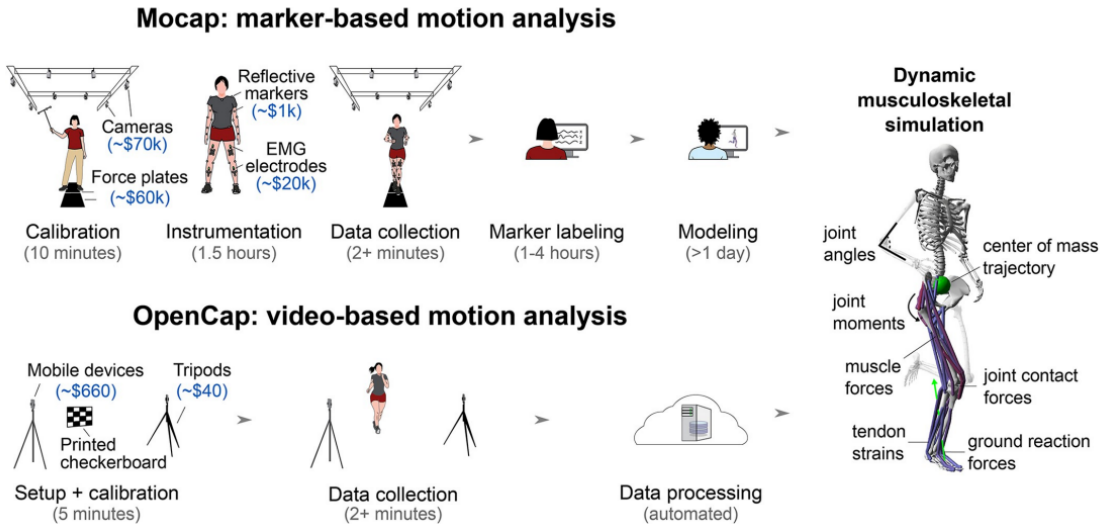


Figure 2.7: Marker-based motion capture vs. OpenCap motion capture [72]

These type of systems, while non intrusive since they only require cameras to record movement, can be expensive depending on the quality of the cameras, but in turn due to the markerless nature of these systems, the need of purchasing markers is removed. As disadvantages, these systems typically require more processing and data treatment beforehand in order to have reliable orientation and position data and the accuracy of the

measurements aren't as precise as its counterpart marker-based solution.

2.2.1.2 Non-Optical Motion Tracking Systems

Non-optical motion tracking systems, as the name implies, don't rely on optical devices or techniques to estimate physical movement. These type of tracking systems instead use different types of sensors to provide such estimation on rigid bodies.

The main types of non-optical motion tracking that are used today are: **mechanical motion tracking systems, magnetic motion tracking systems, acoustic motion tracking systems and inertial measurement units**.

Mechanical motion tracking systems use physical contraptions such as exoskeletons or mechanical rigs that are attached along body segments of a person in order track the physical movement. These mechanical contraptions typically use potentiometers that are placed in body joints and that connect the mechanical rigs placed on the various body segments[47]. Based on the rotation or movement applied to these potentiometers a resistance is produced and results in a change of voltage to the output end of the potentiometer. The output voltage is then measured using an analog-to-digital converter (ADC) which then allows to estimate the position or angle formed by specific body segment[2].

The work from Yeongyu Park et al. [49] illustrates the use of a mechanical motion tracking system for measuring finger motion through the use of linear potentiometers, flexible wires and linear springs. This system consisted on attaching a linear potentiometer with a linear spring connected to a flexible wire that runs along each finger and these were attached atop of a glove as seen in Figure 2.8. As a flexible wire was moved by a finger's motion, the respective joint angles could be calculated through change of length on the flexible wire.

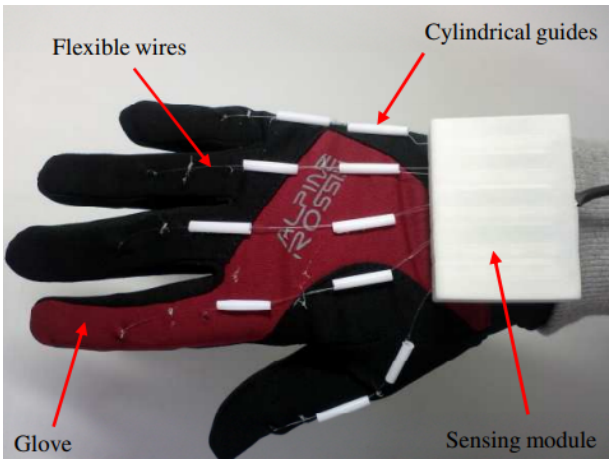
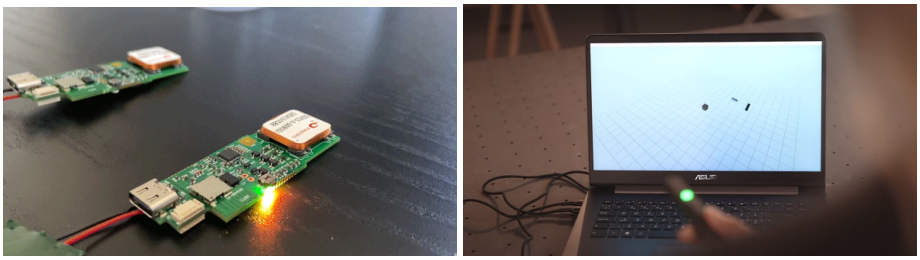


Figure 2.8: Wearable sensing glove prototype [49]

These mechanical motion tracking systems, depending on their implementation, can be more intrusive and less comfortable to wear specially when using solutions based

on exoskeletons. The accuracy can be moderate or high if performed in controlled environments but the costs tend to be very high due to the complexity of the mechanical rigs and their inherent hardware requirements.

Another form of non-optical motion tracking is through the use of **magnetic motion tracking systems** which infer position and orientation through correlation of electromagnetic fields (EMF). The electromagnetic sensors typically are comprised of transmitters and receivers. The transmitters are responsible for emitting magnetic fields while receivers are responsible for capturing said magnetic fields and processing it into reliable information. An example of this is the commercial application ***AMFITRACK*** which uses electromagnetic fields to track absolute position and orientation relative to a transmitter and a receiver component as seen in Figure 2.9, enabling the effective and reliable motion tracking [5]. These systems depending on their complexity can be more or less expensive, in terms of comfort they can be intrusive if the prototypes are more wider and bulkier, and the accuracy of measurements is moderate to high since the measurements can be prone to external magnetic interference and hence producing errors.



(a) (b)

Figure 2.9: AMFITRACK - Electromagnetic motion tracking system. (a) AMFITRACK component chip. (b) Viewing orientation and position from a graphical view. [5]

Acoustic motion tracking systems rely on sound waves to perceive position and movement of rigid bodies. As seen in magnetic system capture, typically there is a transmitter component and one or more receiver components which have microphones or sensors that are able to detect the emitted sound waves [27]. The emitter component emits sound waves at different times and frequencies which are then propagated around the environment and bounced into objects including the receivers. The receivers then capture the reflected sound waves and, based on the transmission delay between the emitter and receiver, the acoustic system can perceive the distance between each other. Using the measured distances and knowing the positions of each receivers we can apply the triangulation method to calculate the position of a tracked object. Advantages of acoustic motion tracking is that it is a non intrusive method, the cost can be affordable depending on the type of setup required but as disadvantage the accuracy might not be the best since these types of systems are prone to be affected by background noise or interference.

Last but not least we have **inertial measurement units** which are electromechanical devices that contain an array of sensors that enable measurement of motion [30]. These

devices are typically composed by a 3-DOF accelerometer which detects the change of velocity through the measurement of linear acceleration and a 3-DOF gyroscope which measures the rotational rates or change in angular velocity along all three axis. More recent inertial measurement units also comprise a 3-DOF magnetometer which measures the strength and direction of the Earth's magnetic field along the three axis.

With the advances in technology, various hardware components that were initially big in size and expensive, started gradually shrinking overtime and proportionally their production costs also reduced. Inertial measurement units were one of these cases where initially they were devices of large scales and also very expensive to acquire, and now they are conceivable in microchip sizes and are far more affordable which in turn made them more available for public use. These small sized inertial measurement units were named as micro-electromechanical system (MEMS) inertial sensors.

Despite inertial measurement units having the aforementioned array of sensors, the raw data obtained from these sensors individually isn't enough to provide orientation and position over IMUs since accelerometers overtime accumulate error due to biases, gyroscopes accumulate drift and magnetometers can be affected by magnetic disturbances [11].

In order to provide accurate estimation of the orientation and position of IMUs, **sensor fusion algorithms** must be applied. These algorithms process and combine the various raw data values from each sensor of an IMU into an attitude and heading reference system (AHRS) which in turn provides orientation and position of the IMU through the roll, pitch and heading values. Also, IMUs typically require prior sensor calibration in most cases in order to guarantee accurate data output and to ensure this output remains the same upon repeating the measurements in different conditions and settings.

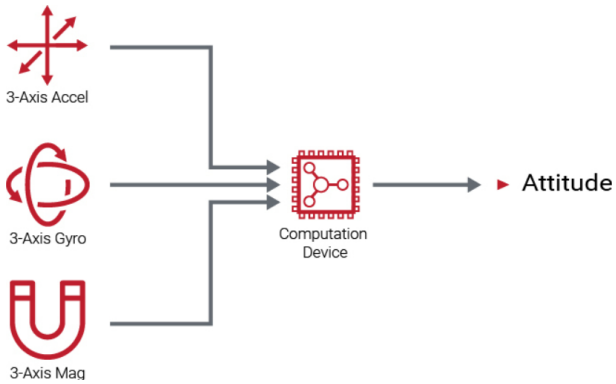


Figure 2.10: AHRS Diagram [11]

MEMS IMUs offer reliable measurement readings due to their array of sensors, are comfortable to wear due to their size and weight, are cheap to acquire making them an excellent and affordable option for providing motion capture and can be easily integrated into development boards to create motion sensing prototypes which is one of the objectives of the proposed work.

After going through the state of the art of motion capture systems and reviewing their advantages and disadvantages we can now summarize our research by considering the three important indicators: **measurement accuracy, comfort and cost** as seen in the Table 2.1.

Motion Tracking System	Measurement Accuracy	Comfort	Cost
Marker-based systems	High	Medium	High
Markerless systems	Medium-High	High	Medium-High
Mechanical systems	Medium-High	Low	Medium-High
Magnetic systems	Medium-High	Medium	High
MEMS Inertial sensors	Medium-High	High	Low-Medium
Acoustic systems	Low-Medium	High	Medium

Table 2.1: Motion tracking systems summarized comparison

2.2.2 IMU sensor fusion and orientation representation systems

When working with inertial sensors, it is important to know how the process of sensor fusion works and what available algorithms we have at our disposal. Also it is imperative that we know which representation system we should use to provide us orientation on the IMUs and which representations to avoid due to their inherent problems.

2.2.2.1 Orientation representation systems

For calculating the knee joint angle, there's a need to first understand how orientation is expressed in mathematical terms.

In this section it is explored how **euler angles, rotation matrices, and quaternions** work for providing orientation in three dimensions.

Euler angles are a three dimensional representation system that decomposes three dimensional rotations into three separate angles denoted by α**(yaw),** β**(pitch) and** γ**(roll)**. There are two different conventions for Euler angles, **classic convention and Tait-Bryan convention**. On classic convention the three elementary rotations are performed only on two axes, meaning that given three rotations, e.g. the first rotation is applied to the x axis, the second is applied to the y axis but the last one would have to be again applied on the x axis [60].

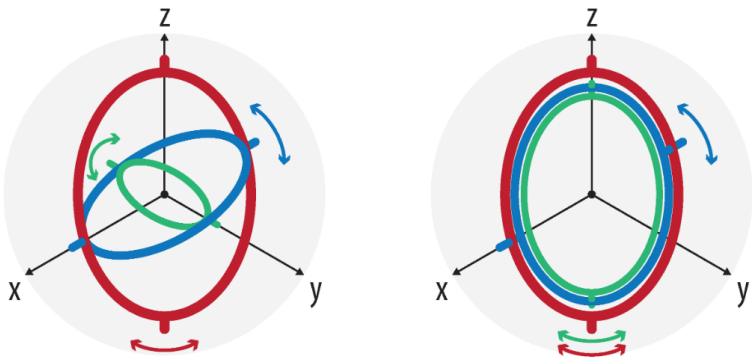
The Tait-Bryan Euler Angle convention on the other hand allows for the three elementary rotations to be made on each respective axis. The configuration of rotations matter since rotating a rigid body by the configuration XYZ is completely different from the rotation configuration ZYX. The elemental rotations can also be intrinsic or extrinsic, meaning that on case of intrinsic rotations if a rotation is applied the coordinate system is also rotated as consequence of the previous rotation operation, while extrinsic rotations are performed on a fixed coordinate system and don't depend on previous operations.

Another way to represent orientation is through **rotation matrices** which consist of 3x3 matrices that represent each axis per row and each of their corresponding row elements defines the resulting direction of each axis after a given rotation has been performed [60]. The three elementary rotations can be either computed individually or aggregated into a single rotation matrix as long as the order is respected as seen in Figure2.11.

$$R = R_z(\alpha)\,R_y(\beta)\,R_x(\gamma) = \underbrace{\begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\text{yaw}} \underbrace{\begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix}}_{\text{pitch}} \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & -\sin\gamma \\ 0 & \sin\gamma & \cos\gamma \end{bmatrix}}_{\text{roll}}$$
$$= \begin{bmatrix} \cos\alpha\cos\beta & \cos\alpha\sin\beta\sin\gamma - \sin\alpha\cos\gamma & \cos\alpha\sin\beta\cos\gamma + \sin\alpha\sin\gamma \\ \sin\alpha\cos\beta & \sin\alpha\sin\beta\sin\gamma + \cos\alpha\cos\gamma & \sin\alpha\sin\beta\cos\gamma - \cos\alpha\sin\gamma \\ -\sin\beta & \cos\beta\sin\gamma & \cos\beta\cos\gamma \end{bmatrix}$$

Figure 2.11: Rotation Matrix Representation [1]

Despite Euler angles being easier to visualize for humans, their data holding three values and also being easier to comprehend due to their elementary rotations, they suffer a problem that can be demonstrated via a gimbal mechanism as depicted in Figure 2.12. When two rotational axis overlap on each other, a singularity occurs and one degree of freedom is lost in the process, this phenomenon is called **Gimbal Lock**. Typically to address these points of singularities requires special treatments and mathematical artifices to correct the angle readings. Additionally Euler angles as previously seen, depend on axis sequences which can differ from each application.



(a) Gimbal mechanism. (b) Singularity of mechanism.

Figure 2.12: Gimbal Lock - when two rotational axis overlap on each other a degree of freedom is lost [45]

Due to these problems, a better and more reliable way is needed to represent orientation and that way is through the use of **quaternions**.

A rotation quaternion is a four dimension [45] or hypercomplex number that can represent accurately the orientation of a rigid body in three dimensions and it is mathematically represented as:

$$q = q_w + q_x + q_y + q_z$$

Where $q_x, q_y, q_z, q_w \in \mathbb{R}$ and q_x, q_y, q_z form the unit vector part and q_w the scalar part. Quaternions are non commutative meaning that the order of multiplication matters, allows back and forth conversion to Euler angles and rotation matrices and don't suffer from gimbal lock phenomenon [59] making them ideal for representing three dimensional rotations.

Some important quaternion properties that will be used on the developed work should be addressed here, specifically:

- **All rotation quaternions must have their norm,** $|q| = \sqrt{q_w^2 + q_x^2 + q_y^2 + q_z^2} = 1$
- **From the norm formula we also have that,** $q_w = \sqrt{1 - (qx^2 + qy^2 + qz^2)}$
- **The conjugate of a quaternion is represent by,** $q^* = (q_w, -q_x, -q_y, -q_z)$
- **The conjugate of a unit quaternion is equal to the inverse of that quaternion,**

$$q^*_{unit} = q^{-1}_{unit}$$

- **The multiplication of two quaternions** $q1 \times q2$ **is equal to:**

$$\begin{aligned} & q1 * q2 \\ & = (q1_w, q1_x, q1_y, q1_z) * (q2_w, q2_x, q2_y, q2_z) \\ & = \begin{pmatrix} (q1_w q2_w) - (q1_x q2_x) - (q1_y q2_y) - (q1_z q2_z) \\ (q1_w q2_x) + (q1_x q2_w) + (q1_y q2_z) - (q1_z q2_y) \\ (q1_w q2_y) - (q1_x q2_z) + (q1_y q2_w) + (q1_z q2_x) \\ (q1_w q2_z) + (q1_x q2_y) - (q1_y q2_x) + (q1_z q2_w) \end{pmatrix} \end{aligned}$$

These properties later on the developed work will be vital for calculating the relative angle between two quaternion orientations which will be the method that will be employed to infer the knee joint angle.

In order to have more accurate and reliable measurements from motion sensors and to obtain orientation we must employ **sensor fusion algorithms** to integrate the various sensor values into a more rich and reliable data format that is consistent throughout the wearable sensors usage.

There are several sensor fusion algorithms as of today, some worthy mentions are the **Complementary Filter, Kalman Filter, Mahony Filter and Madgwick Filter**. The study of each sensor algorithm is out of scope for this thesis and we will only briefly go through the various features that selected sensor fusion algorithm used for the developed work **TDK Invensense's ICM-20948 digital motion processor (DMP)** provides and will not go very in-depth on how the sensor fusion algorithm works since its documentation is closed source.

The **digital motion processor** is an onboard chip within the ICM-20948 that offloads the computation of motion processing algorithms from the microcontroller to the sensor's chip [65]. By offloading the motion processing algorithms to the DMP instead of the microcontroller, the microcontroller can focus its resources and computational power for other tasks, thereby reducing its power usage. The DMP also provides automatic runtime

and background calibration on accelerometer, gyroscope and compass, eliminating the need for users to explicitly calibrate each sensor, which in turn simplifies the development process. The DMP collects the data from the various sensors, processes the data using a selected sensor fusion algorithm, and then stores the resulting processed data in the FIFO, so it can be read and used for the application's purpose.

Despite various sensor modes being available on the DMP, the developed work will solely focus on modes that offer quaternion values, specifically:

INV_ICM20948_SENSOR_GAME_ROTATION_VECTOR - provides a 32-bit 6-axis (accelerometer and gyroscope) quaternion value;

INV_ICM20948_SENSOR_ORIENTATION - provides a 32-bit 9-axis (accelerometer, gyroscope and magnetometer) quaternion value and also heading accuracy;

Due to these modes offering quaternion values, we can experiment and extrapolate the knee angle through quaternion operations and also we can validate the accuracy and results using the two sensor modes.

In the developed work we could've also studied and experimented with other Arduino libraries, but since the emphasis of this thesis was to use these new IMU sensors that offload computation to their onboard chips, this experimentation was discarded.

2.3 Knee monitoring systems and knee angle joint estimation techniques

Another form of knee joint monitoring system was explored by Kobashi et. al using a force and magnetic, angular rate, and gravity sensor (MARG) [41]. A MARG sensor encompasses an magnetometer, gyroscope and accelerometer similar to an inertial measurement unit (IMU). The knee monitoring system was accomplished through two MARG sensors (one on the thigh surface just above the patella, another bellow the patella on the shank), a pressure sensor embedded into the shoes and lastly a Bluetooth unit located in the lower back which relayed all the sensors data to a processing system that consolidated and transformed the data into a three degree of freedom knee joint angle estimations as seen in Figure 2.13. Due to both sensors having their local coordinate system, the expected acceleration vector for the knee is estimated by the difference of the two acceleration vectors which converge to knee joint center.

Saba Bakhshi et al. also investigated three forms of measuring the knee joint angle. The first attempt on measuring knee joint angle was through a wearable brace with an absolute magnetic kit encoder placed on the femoral shaft [38]. Different pulse widths symbolizing different angles would be generated by the encoder, and through the CSMB12 microcontroller unit these pulses would then be counted and added up using the microcontrollers accumulator.

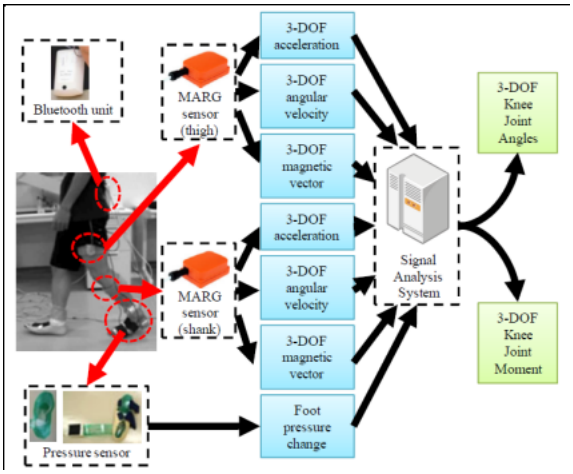


Figure 2.13: Kobashis knee monitoring system [41]

These angle readings would be then transmitted from the microcontroller to a smartphone using a Bluetooth module BlueSMiRF Gold from Sparkfun. The maximum, minimum and current knee angle was displayed on the smartphone and also on LCD that was paired to the microcontroller. To begin the motion tracking of a patient, the patient would have to press an on/off switch on the microcontroller, afterwards, then pressing a start button to start recording and emitting data, and lastly, on the patient's smartphone, connected to the sensor by Bluetooth, the values would start being displayed on the screen. According to Saba's results, this method of measurement wasn't the most optimal since after every exercise, the patient would have to adjust the brace in the correct location, otherwise the measured angles would be inaccurate. Another disadvantage of this method was the discomfort of wearing the heavy brace for long periods of time.

The second approach Saba et al. tried was to measure the inner angle of the knee joint using two flex sensors and elastic supportive cloth [38]. These flex sensors were mounted underneath the knee in order to provide flexion and extension to be made freely. This system measured the inner knee angle through the measurement of resistance on a current source provided by bipolar junction transistors that were connected to the flex sensors. As the knee performs more flexion, more flexion is applied to the sensor and consequently more resistance is applied to the current. This resistivity created from flexion would be then converted to voltage by the constant current source. The measured voltage from the microcontroller board served as input to the extended Kalman filter and fusion system, which as result provided the estimation of the knee joint angle. In this system, 180 degrees is when the knee is fully extended, while the increase of flexion reduces the angle.

The third approach Saba et al. used for measuring the knee joint angle was through the use of two inertial motion unit sensors (IMUs) [38]. An IMU is typically comprised of an triple axis accelerometer, a three degrees of freedom gyroscope and magnetometer, but the accelerometers and gyroscopes are most commonly used for inertial measurement. The Euler angle measurements provided by the IMUs are then transmitted via Bluetooth

to a computer for later data analysis, and the angle estimation is calculated in real time via LabView.

The IMUs were mounted in the front of the shank and thigh with straps to hold them in place. For the actual angle estimation, the IMUs had to be placed on a flat surface parallel to ground in order to calibrate to the same zero reference. The accelerometer was used to find the flexion angles and the gyroscope to eliminate vibrations on the accelerometer. The IMU placed on the thigh, determines the absolute thigh angle which is formed by the gravity vector and a perpendicular vector to the femur. The IMU placed on the shank would determine the absolute shank angle which is also formed by the gravity vector and a perpendicular vector to the tibia. The difference between the thigh and the shank angles would then result in the expected estimation of the knee joint angle. To validate the angle estimation from the IMUs, Saba et al. placed a series of reflective spheres along with the IMUs, which acted like Helen-Hayes markers to do infrared motion capture of the lower body.

Despite the two systems having different sampling rates, with the infrared having 100 Hz and the IMUs having 5 Hz, as long as the test subject would remain still at the beginning and at the end of the test trial, the data retrieved from both systems would then synchronize. The results depicted in Figure 2.14 were that IMU knee angle calculation would approximate to infrared knee angle calculation on the exercises that were performed: knee extension while seated, sitting down and standing up, moving the knee up and down while sitting, lastly combined movement patterns of gait and squats.

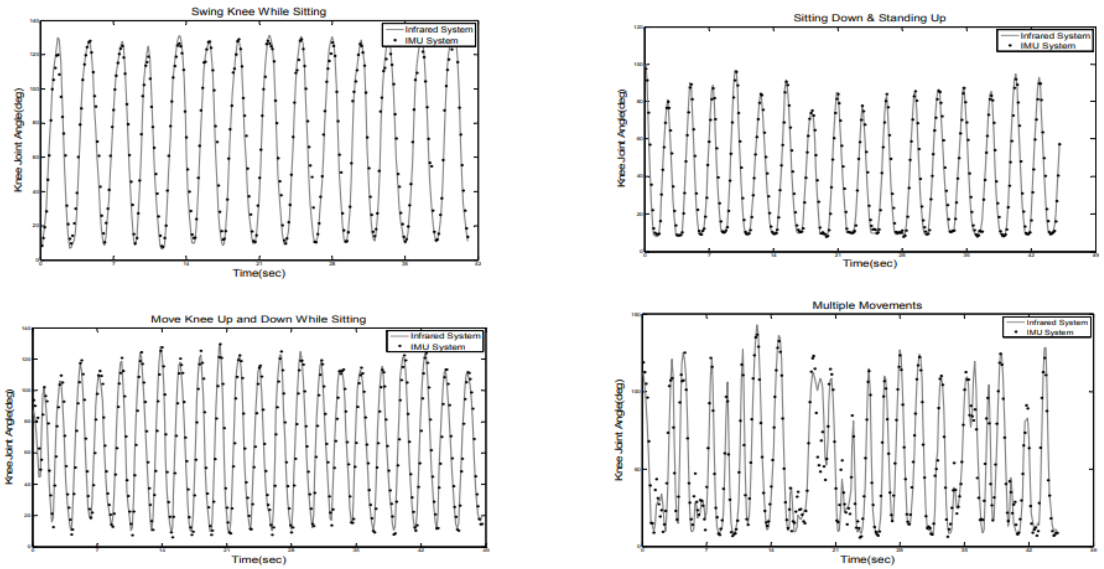


Figure 2.14: Measurement accuracy results between IMU and Infrared systems [38]

Antunes R. et al. also achieved knee joint angle measurement through inertial motion units by using MotionSense wearable sensors [9]. The sensors consisted on two sensor nodes attached on the lateral sides of the thigh and shank using a dual patch system as

seen in Figure 2.15. The measurements done by the sensors, were then communicated via Bluetooth to a mobile app installed on the patient's phone.

Each sensor was comprised of an IMU with six DOF with a sample rate of 50Hz, these samples were processed afterwards using a Madgwick filter to calculate the corresponding pitch and roll angle of each sensor node. The dual patch system consisted of one patch being in contact with the skin for longer periods of time and another patch that adhered to the first patch and which was removable, to provide sensor charging. After the chargeable sensor patch is removed, the sensor-to-leg calibration is lost and must be re-calibrated. The knee joint angle is calculated via the angle formed between the tibial mechanical axis and the femoral mechanical axis. Despite being unable to measure the relative orientation of the mechanical axes, this orientation is approximated by the sensors placed on the two leg segments through the misalignment angles, which are the difference between the anatomical mechanical axes and the sensor axes. Hence the following relationship:

The calculation for the misalignment angle

$$\delta_{\text{Knee}} = \delta_{\text{Femur}} - \delta_{\text{Tibia}}$$

The calculation for the knee joint angle

$$\theta_{\text{Knee}} = \theta_{\text{Femur}} - \theta_{\text{Tibia}} - \delta_{\text{Knee}}$$

After the patient removed the rechargeable patch, the δ_{Knee} had to be recalculated by the patient resting his foot on a rigid box of 10cm. This ensures the re-calibration of the relative orientation of the mechanical axes.

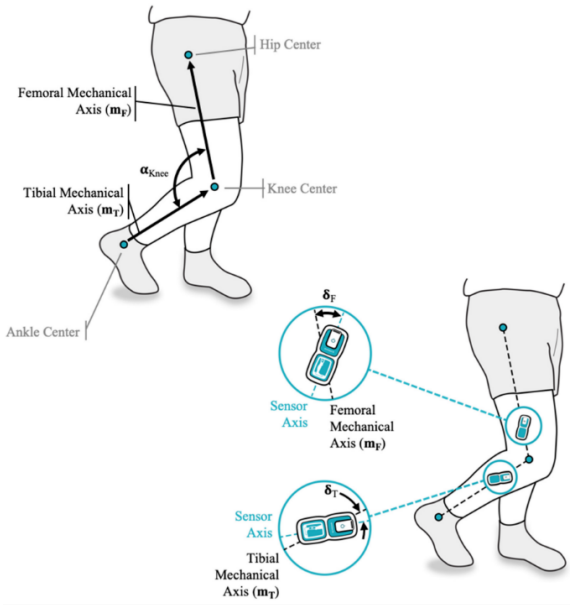


Figure 2.15: Calculation for knee joint angle in MotionSense Sensors [9]

To validate the measurements provided from the MotionSense sensors, Antunes R. et al. used video motion capture during a series of various exercises recording the patients

sagittal plane. The video motion capture comprised a camera to record imaging and machine learning pose estimation library called OpenPose to determine the hip, knee and ankle joint locations provided by the video feed. This library was susceptible to camera angle, body shape, color and looseness of the patients clothing meaning that during trials, the results produced by OpenPose had to be visually inspected to guarantee that the joint centers were correctly identified. On each video frame, the knee angle was estimated based on inner angle formed from the mechanical axis of the femur and tibia. Any patient on which the OpenPose would wrongly identify the joint centers, this patient would be discarded from the comparison. To synchronize the measurements between systems, an sample offset was applied to maximize the covariance between video and sensor measurements.

This method revealed some limitations, since the method relied on an accurate reading of the goniometer with the foot on the box, on first visit to the health professional in order to compare to future readings from the sensors. If this reading was wrong, it could result in sub optimal angle estimations. Additionally the trials were made on young adults, the knee range of motion differs depending on the target population that is being tested.

Bryan Rivera et al. developed an method to infer the type of user activity that a subject was performing (standing, sitting or walking), based on the knee angle and angular speeds [58]. This was accomplished through the use of a developed 3D printed knee goniometer (see Figure 2.16) and an algorithm.

The developed wearable motion-sensor was aimed for one degree of freedom (extension and flexion), and was comprised of two arms, one in the shank and another in the thigh. To hold the wearable in place, Velcro combined with straps and elastic bands was used to hold the arms in place and to maintain them properly aligned to the extension/flexion axis which was on the lateral side of the leg. The arms were 3D-Printed, and inside the shank's arm resided the electronic components and the hall effect sensor. The measurement of the joint angles were through the magnetic rotatory encoder provided by the hall effect sensor (model AS5048B), which the measurement itself was then processed by a microcontroller Atmega328 and finally sent the processed data via Bluetooth with a sample rate of 100Hz to an Android device or to an computer. The mobile application would serve as initial calibration of the sensor, in order to set which knee was getting measured and to define the starting zero position.

Figure 2.16: 3D printed knee goniometer [58]

The activity detection algorithm was based on a finite state machine which used the angular position and the angular speed of the knee joint to infer which position the subject was performing.

The state machine was composed of 5 states, where each of these states represented a different situation.

State 0 was used to represent no subject motion, while on going the other states meant the user was doing some sort of activity. To evaluate if the subject was standing or sitting, it would requires to be in the state 0 initially. During state 0, if the measured angles were below 20 degrees, then the subject would be in the standing position, if the angle was more than 70 degrees then the user would be sitting. For infering these positions, the angular speed wasn't taken into account. On the other hand, on the remaining states, the peaks of knee flexion angle during gait cycle were being analyzed.

One cycle of walking activity would represent four concavities, two up and two down, these peaks on the concavities represented the peaks on extension and flexion. With the concavities representing angles and by evaluating the sign of the angular speed, if the sign of the angular speed was positive, this meant that the angle measured was increasing. On the latter side, if the sign of the angular speed was negative, this meant the angle measured angle was decreasing. To detect if there was walking activity the maximum and minimum angles were evaluated during the sequence of the remainder states (state 1, 2, 3, 4). The respective sequence angles for each state would be comprised in the ranges of [30º ; 70º], [-5º ; 15º], [0º ; 30º] and [-10º ; 15º] to complete a cycle of walking.

The validity of measurements was issued against a commercial goniometer and against the measurements provided by an Aktos-t IMU system while subjects walked at three different speeds wearing both systems.

Almost no absolute mean error was detected when comparing with a commercial goniometer but when assessing with the Aktos-t system it was concluded that the evaluated mean absolute error varied based on which subject was wearing the sensor.

2.4 Remote Physiotherapy System Design

After going through some of the existing solutions on knee monitoring and rehabilitation systems, and exploring their different implementations, challenges and architectures, in this sesction, some remote health care system design guidelines are analyzed. It will also be explored the advantages and trade offs of using different types of technologies and frameworks that can be applied to the *KNEEMOR* system. In this section it is addressed the technical choices that led to the creation the *KNEEMOR* system.

2.4.1 Remote health care system design guidelines

When conceiving remote rehabilitation systems, these should comply to a minimum baseline of requirements that follow the industry best practices.

The work from L. Ismail et al. [35] identified that health care system designs should follow 7 requirements: **medical record data, real-time data access, patient participation, data sharing, data security, patient identity privacy and public insights**.

The **medical record data** requirement is related to patient record keeping, i.e. the way that patient records are kept and managed. For these types of systems different ways of accessing such records should be available to patients. The patient records are typically accessed and viewed through an application and additionally can be exported in a report format for later review or access.

Real-time data access is also an important factor for health care systems, since it provides both patients and health professionals with the capability of delivering and receiving real-time patient care. This entails communication, appointment management and exercise prescriptions.

Also, communication between patient and health professional plays a pivotal role for enabling telehealth systems. As seen in previous chapters, having remote solutions that enable interaction between patients and health professionals can be beneficial both in terms of operational costs and also in terms of geographical constraints for accessing health care [61].

Patient participation requirement entails that patients should be able to interact with health care systems to check and track their health progress, schedule appointments or request medical assistance. These systems should be intuitive for patients and health professionals alike in order to drive better engagement and adoption for both stakeholders. IoT based health care monitoring using wearable motion-sensors plays an important part in this requirement, since through continuous monitoring of a patient's health condition, health professionals can react faster to deteriorating health states that a patient might be experiencing.

The **data sharing** requirement describes that medical record sharing should be possible throughout various health care providers. This makes sense for correlating diagnostics and to provide a wider pool of medical history resources that can be useful for identifying pathologies and diseases on other patients. But this requirement is not trivial to implement, since different privacy and data regulations are applied to different countries and also due to health care systems not providing a common standardized interoperability for sharing their resources.

Security and privacy are also important requirements, since secrecy of the data should be ensured for all patient medical data and underlying systems should adhere to the best security practices to minimize the attack vector for hackers and vulnerabilities.

In the developed work, some of these requirements will be addressed while others will be partially implemented or out of scope due to the time constraints and overall complexity of the current system.

For conceiving a remote health monitoring system, four main processes must be addressed: **data acquisition** (through sensory measurements, images, etc), **data storage**, **data transmission** (through some sort of network and protocol) and lastly a **feedback**

system (in order to inform the patient of his condition or to correct him, through visual or audible feedback).

Fatih's A. work [20] proposed an architecture for a e-rehabilitation system that comprised partially of the processes mentioned above.

This architecture had a sensor layer where the assessed joint angle measurements were retrieved from the respective sensors and parsed on the microcontroller to remove any drifts. A server layer which resides typically on the cloud provides the data exchange and storage for the sensors gateway and also the client application, and finally the client layer where the end users in this case health professionals access and evaluate the patient's data, contact them to provide them with support or even parameterize exercises for them.

A similar approach to Fatih's proposal was Yung-Chung's architecture [7], but the difference in his approach was that he relied on a message broker using the MQTT protocol to relay the data from the sensors to the database and additionally to an user interface that was accessible via a magnetic card.

Both approaches are good implementations, with the first one providing a direct transmission between the server and the sensor and the second implementation having an intermediary broker to relay the data of the sensors, however both of these approaches require a WiFi connection so that the data can be transmitted which can be a disadvantage. During the early development of the proposed work, MQTT was considered but later on was discarded for this exact fact, and has been replaced since then by BLE as a means of transmitting the sensor data.

2.4.2 Webapps and Mobile development

When developing remote health care systems one of the many important aspects to consider is the user interfaces (UIs) that these systems provide and how useful and practical these are for both parties involved: patients and health practitioners.

Also depending on the target audience (patients or health practitioners), the underlying user interfaces should be refined and accommodated. For health practitioners mobile apps might be less appealing for their everyday operations due to their small screen size, making it less practical for health professionals to swiftly fill out batches of reports, prescribe treatments, schedule and approve appointments and perform other medical related tasks. On the other hand, patients might prefer a mobile application to quickly access their medical data and to view such information from the comfort of their phone.

In order to make a frontend design decision based on each stakeholder, an evaluation over the today's frontend development approaches is needed: **web applications, native mobile applications and hybrid mobile applications**.

Web applications or web apps are essentially websites that provide rich, interactive interfaces and that are responsive on various screen sizes [75]. These websites are normally built using standard HTML, CSS and Javascript, but can also often recur to Javascript frameworks like **React, Angular and Vue** which accelerate the development process and

provide a wide a array of libraries for building user interfaces. Web apps are platform-agnostic, typically relying only on a browser to serve their content. This approach usually involves only one codebase, unlike its counterpart the native mobile applications, which must be tailored in different programming languages to run on different operating systems. However, web apps are usually internet dependent and typically cannot provide native features that mobile apps offer.

Native mobile applications in turn provide native functionalities like camera access, GPS and location services, calendar and organizer, file manager, notification centers and much more. This approach allows for offline access to its resources, provides a higher degree of security since applications need to be approved on their respective app stores, supports push notifications out of the box which helps drive better user engagement, and typically offers faster performance than web apps in some cases. However, developing native mobile apps comes with a set of disadvantages, especially the time consuming process of tailoring and developing an application in different codebases **(Java, Kotlin and Swift)** and also for different operating systems, namely Android and iOS. Maintenance can also be expensive since it requires managing two applications instead of one and additionally when shipping native mobile applications a fee must be paid and the application must be approved by their respective independent stores. Finally native mobile applications require device storage in order to function properly.

Progressive web applications (PWA) are an extension from traditional web apps that provides offline capabilities and support to web applications. It offers reliable and fast load speeds regardless of network conditions due to their underling caching capabilities, and are installable like native mobile applications while remaining platform-agnostic. These installable web apps are essentially standalone websites installed on the user's devices, this is made possible through the PWA service workers assistance and with the definition of the manifest file, which has the metadata related to the progressive web app [75]. This approach also provides some native capabilities like camera access and push notifications similar to mobile apps, but these features are generally more limited. A strong advantage of PWAs is that they do not require app stores for their installation, instead it only requires a browser on the user's device and PWA support from the web application. However, allowing this freedom for installing an application without the use of an app store increases the risk of security exposure on the user's device. Another disadvantage is the adoption of PWAs on mobile devices being slow, despite being gradually adopted over the years on Android, there is still a lot of friction on iOS adoption. This frontend design approach is particularly interesting for health practitioners since it allows them to leverage medical applications on both computers and mobile phones, making it a versatile option for their everyday operations.

For the developed work, this will be the selected approach for building the physiotherapist's user interface for the system and the library that will be used for building said interface will be React.

React is a library for building web interfaces based on the Javascript language developed

by Meta [52]. The main premise of React is building user interfaces through the use of **components**, which are basically individual building blocks that when combined and structured make up a user interface and pages are created. For building said components a special syntax is used named **Javascript XML (JSX)**. This syntax allows for creating custom markup that looks and feels like HTML but in fact under the hood is just plain Javascript which renders content. Components can be **functional or class based**, but as of today the functional approach is the standard for developing React applications.

A particularity about React is the way the library handles state update changes. A copy of the **Document Object Model (DOM)** named **Virtual Document Object Model (VDOM)** is maintained in memory by React in order to compute the changes that are required on the real DOM [74]. This process of computing the necessary UI changes is named **reconciliation** and can be broken down into 3 stages: **the state change, the diffing algorithm and the re-render**. To explain this phenomenon, an example will be explained through the depiction of the Figure 2.17. React, upon receiving a state update on the component C, triggers a re-render of that component. For the preparation of the re-render, a copy of the DOM is issued, generating a VDOM and by which React reflects the new state update on the VDOM Tree. Afterwards, the diffing algorithm executes which determines the minimal set of changes that are required for the new render, in this case two extra components F and G must be re-rendered accordingly. Lastly the virtual DOM changes become reflected on the real DOM, finishing the re-render.

This feature removes the need for React to do a full page reload but instead does a partial page reload only on the specific components that have suffered state updates. Also React uses a one-way data flow, which means that only parent components can propagate state into child components but the other way around isn't possible. The way to provide two-way data flow in React is by passing state handler functions from parent components into the child components using prop-drilling, and through this we are able to mutate state on the parent component. Managing state this way starts becoming a challenge as a application becomes larger, so to solve this issue, a global state manager like **Redux** is used.

Redux allows for centralizing an application's state in a store, that can be mutated through dispatch actions and queried using state selectors. By having the application state centralized, there's no need of using prop-drilling to access parent state, we can simply define state slices for the state we want to store and query, and instead use a selector or a dispatch action to perform actions on the specific component[2]. Alongside React for providing type safety on Javascript, the developed work will use Typescript.

Typescript is a statically typed language extension of Javascript which provides type safety, better type inference and better code robustness avoiding pitfalls and common issues that Javascript as a dynamically typed language has [71]. By typing our variables and components in Typescript, the inputs and outputs of our application are ensured,

[2]Redux - Global State Management library - https://redux.js.org/introduction/getting-started

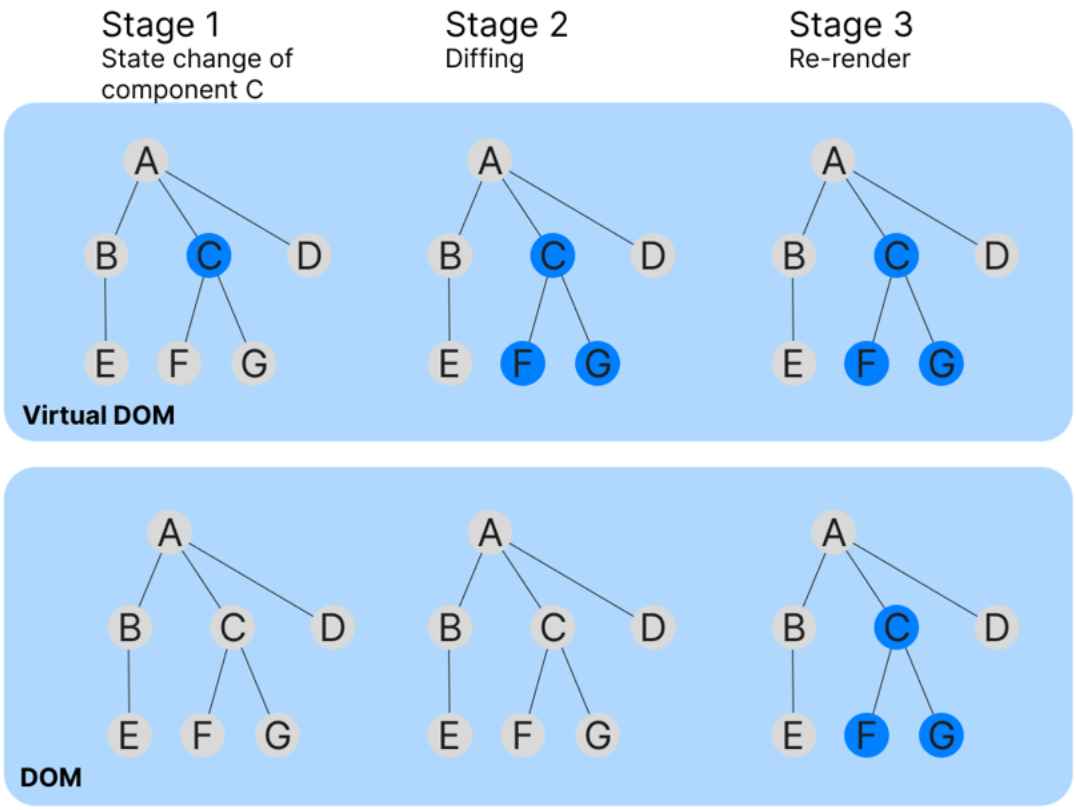


Figure 2.17: React - How state changes are handled in Virtual DOM [74]

which then minimize the potential unwanted side effects that normal Javascript might bring. Also on the developed work, for prototyping our React components fast and styling them accordingly, Tailwind CSS will be used.

Tailwind CSS is a utility first CSS framework, meaning that it provides a set of minimal utility CSS classes that can be combined together for creating custom designs [68]. Unlike typical component-based CSS frameworks **(Bootstrap, MaterialUI and others)** that provide a set of pre-built UI components that can be styled accordingly using their designated APIs, a utility based framework like TailwindCSS does not provide pre-built components. Instead, it only provides a range of CSS classes that can be applied using inline styling on our components. By using inline styling, we remove the necessity of having to manage various CSS files for each component or having CSS conflicts between other components.

Hybrid mobile applications remove the headache of having multiple codebases when developing an mobile application whilst still providing a degree of native capabilities. Hybrid mobile application frameworks like **React Native, Flutter, Ionic, Cordova and Xamarin** allow for development of mobile applications on both Android and iOS using a single codebase easing the time consumption disadvantage from native mobile application development. The native features from mobile devices are accessed typically through a middleware layer or bridge that links the native code into the respective hybrid framework

code. However, by having this middleware layer responsible for the communication with native capabilities, the performance of these hybrid mobile applications is hindered compared to their native counterparts since to interact with native resources, the hybrid applications must flow their requests through the respective middleware which in turn adds to processing delays.

Depending on the type of application that is required, this impacts the choice of which approach to follow, a native or a hybrid approach. If an application is geared towards performance, then a native application approach should be adopted. But, in most cases the hybrid mobile applications are the best choice in general, since they are easier to develop and prototype with, easier to maintain due to not having multiple codebases, offer almost all native capabilities with a slight performance impact and some frameworks also have a great community adoption which promotes the construction of libraries and features that ease the overall development of applications.

Denis Eleskovic on his work [22] made a summarized comparative analysis between hybrid application development on Apache Cordova and a progressive web application created with React library and described their major strengths and weaknesses.

Based on this analysis and also after going through the advantages and disadvantages of each development approach, the comparative analysis from Denis was extended to also incorporate the native mobile application advantages and weaknesses as seen in Table 2.2.

From this comparative analysis it's proven that Native development offers better native module coverage and also better performance but at a cost of development productivity since it's necessary to maintain multiple code bases. Hybrid applications are a middle term in performance and in native module access, but has in turn better development experience and has lower maintenance costs. Lastly, the PWA's suffer from lack of platform and browser support, it's the approach with less coverage of native modules available and also for applications that require native modules is the less performant, but in contrast to these disadvantages, it's the easier approach to implement applications and the approach that is independent of AppStores.

On the developed work, **React Native** will be used for building the user interface for the patient's functionalities and also for providing integration with the sensor layer through **the native BLE module**.

React Native is a hybrid mobile development framework based on the React library, it allows developers to use JSX and components to create native mobile applications. It offers a set of core agnostic native components that can be used interchangeably both on Android and iOS. When compiling said native components, depending on the target operating system that the application is being compiled to, the native components are mapped accordingly to the corresponding native component of that operating system [53].

As previously mentioned, React Native for being an hybrid mobile development framework, accesses the native functionalities from the devices via a Javascript bridge which connects to the target native programming language (Java, Kotlin or Swift). When

Development Method	Strengths	Weaknesses
Native	• Most performant approach • Offline capabilities • Access to all native modules • Direct access to native resources • Consistent user interfaces	• Multiple code bases • Requires App Stores to download • Higher development and maintenance costs
Hybrid	• Unified code base • Offline capabilities • Multi-platform deployment • Access to some native modules • Lower development and maintenance costs	• Requires App Stores to download • Communication to native resources using Javascript middleware bridge • Less native modules access • Inconsistent user interfaces • Semi performant
PWA	• Single code base • Offline capabilities • Independent of AppStores • Lower development and maintenance costs • Easier to implement	• Very limited access to native modules • Unavailable in AppStores • Least performant • Limited platform and browser support

Table 2.2: Strengths and weaknesses between Native, Hybrid and PWA applications, Table extended from [22]

building mobile applications on React Native we have two routes for creating a project: **a barebone React Native project or a React Native Expo project.** For the developed work, despite being easier to develop the React Native application using **Expo** framework, some native libraries were not supported on Expo at the time of the developed work (BLE to be precise) and due to this constraint a barebone React Native project was created instead.

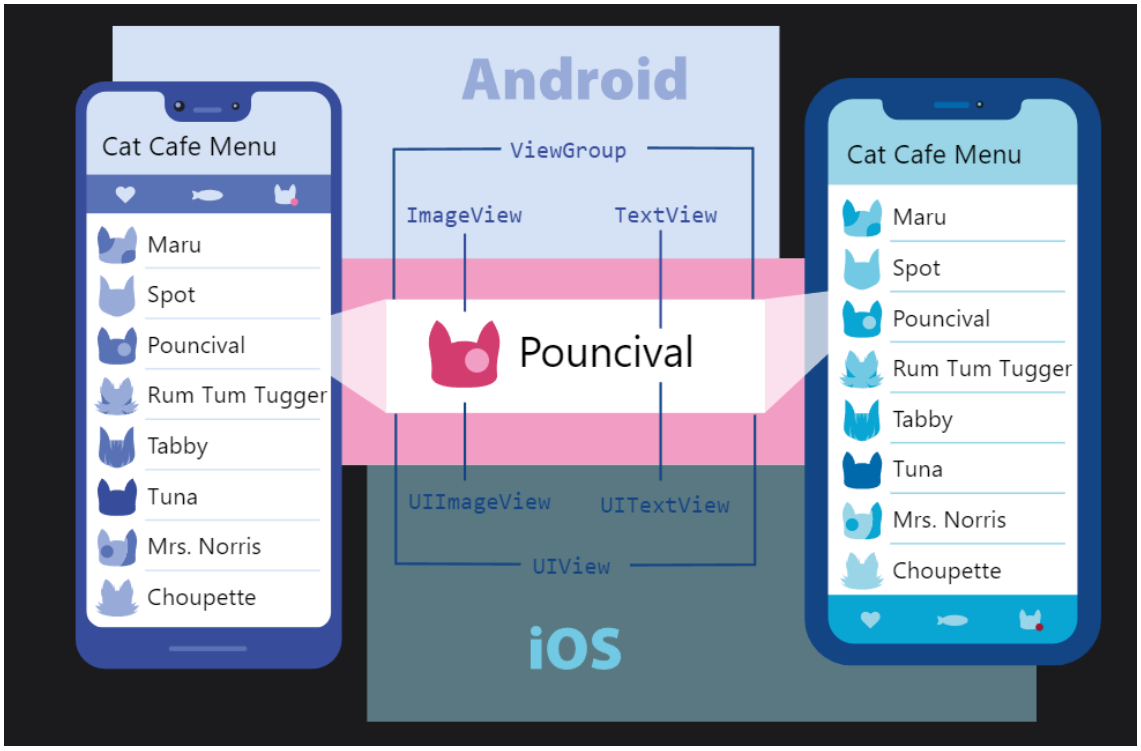


Figure 2.18: React Native - Native components and their subjacent mappings on the respective mobile operating systems [53]

The reason for choosing React Native over other hybrid mobile development frameworks was due to the inherent synergy between React and React Native. While building the patient application or while building physiotherapist application, their respective components could be easily adapted between each project with relative ease. Also due to both options supporting Typescript and TailwindCSS (in React Native the integration is named as **Nativewind**[3]) it makes a solid choice for opting for this framework for the developed work.

Despite being possible to ship an mobile application for both Android and iOS, via React Native, the scope of this thesis will solely be on Android, due to time constraints and lacking the necessary equipment to compile and test on a iOS device.

2.4.3 Backend development and software architecture

Defining a robust backend development stack is another crucial point when designing a health care system since these types of systems are critical and typically must have minimal or no downtime at all. Choosing a backend development stack entails electing the most suited backend programming language, selecting the appropriate database that fits this use case for storing and querying data and also entails choosing the right software architecture for developing the underlying functionalities and services that the backend layer will provide.

[3]Nativewind - Tailwind CSS on React Native - https://www.nativewind.dev/

When choosing a backend programming language, various factors that are relevant for the developed work must be taken into account, i.e. **development experience, maturity of the language and its ecosystems, security, service integration and scalability** while also following a subset of previously defined guidelines that health care systems should comply to.

Modern backend programming languages offer **frameworks** that provide a wide array of pre-built functionalities that help ease the overall development process of backend applications. Authentication, authorization, 3rd-party integrations, database connection drivers and object relation-mappers, real-time communication protocols, project scaffolding for different types of use cases, security and many others, all of these services are examples of functionalities that are baked into a framework's ecosystem, which in turn allows fast prototyping for developers and removes the need to reinvent the wheel, since these services are well designed and tested, leaving the focus of development only on the product we are trying to build. Some examples of mature and well adopted backend programming languages are **C#, Java, Python and Node.js**. These languages, despite having different syntaxes and being statically or dynamically typed, all provide their own frameworks to accelerate backend development.

For the developed work, the **C# programming language** was chosen alongside its adjacent framework **ASP.NET Core using .NET version 7** for developing the backend application of the system.

.NET is a open source application platform developed by Microsoft that aims to deliver cross-platform support, security, reliability and high-performance [1]. It has great community support providing a vast ecosystem of libraries, is well adopted in corporate environments, provides various project templates depending on the specific application requirements (web, mobile, desktop, micro-services, cloud, machine learning, IoT and game development) and shares a similar flavour to Java in terms of syntax but with reduced verbosity in the language.

Unlike its old predecessor **.NET Framework** which is closed source and didn't have cross platform support, over the years C# has evolved and thanks to the creation of the **CLR (Common Language Runtime)**, cross-platform support was made possible.

The CLR provides managed code execution which entails automatic memory management, cross-language integration, exception handling, security enhancements and other services [18]. The compilation process of .NET applications using the CLR is similar to how Java based languages are compiled, since it relies on an intermediate language named CIL (common intermediate language) that is later on compiled into the operating system's native code. When a .NET application is compiled, its source code is first translated into CIL language and afterwards at execution time a JIT (just-in-time) compiler kicks in and translates the respective CIL code into the respective native code. The CLR environment can be viewed as contained virtual machine on which the .NET code is executed, and through this virtualized environment, .NET applications can run on different operating systems agnostically as long as the underlying operating systems have the environment

installed.

ASP.NET Core is the web framework for developing web APIs and web applications on .NET. The framework provides **DI (dependency injection)** support and comprises a request handling pipeline capable of supporting multiple **middleware** definitions which in turn can perform actions upon incoming HTTP requests before reaching their final destination which is the intended endpoint as depicted in Figure 2.19.

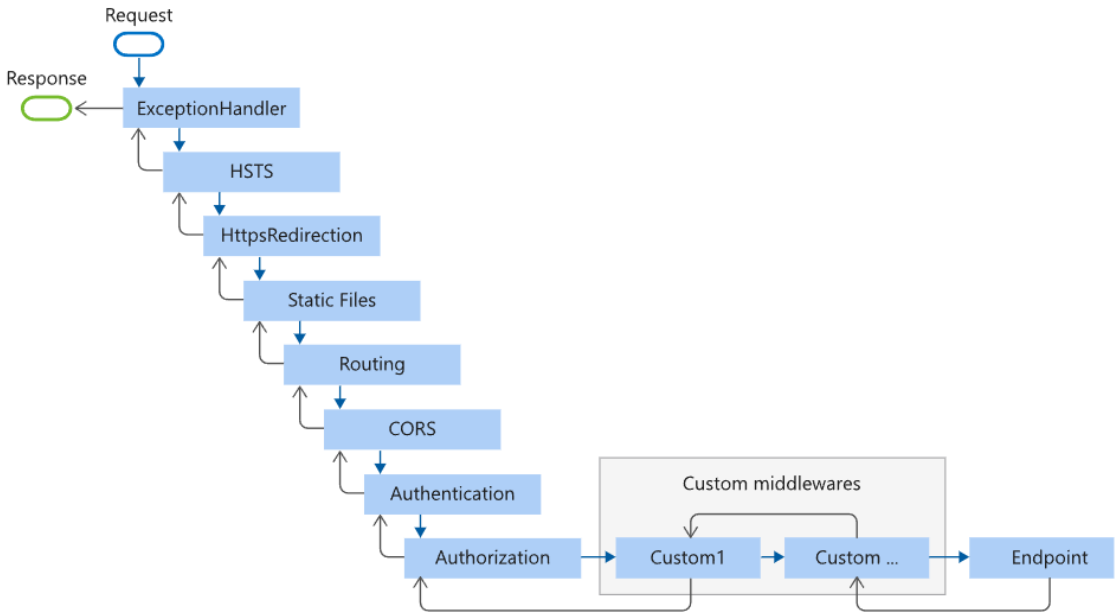


Figure 2.19: ASP.NET Core - Request processing pipeline [10]

For the developed work a backend API will be built so that respective frontend applications can interact with. When building a backend API in ASP.NET Core, a decision must be made between a controller-based or a minimal API approach. The main difference between each approach is that minimal APIs are more light-weighted due to only requiring an endpoint definition and a respective handler function to serve the incoming HTTP request unlike the controller-based counterpart that requires a well defined Controller class to address each API endpoint.

When building the backend API, specific software patterns were followed in order to achieve an organized and robust backend structure. The architecture pattern used in the developed work is the **Clean Architecture pattern** proposed by Robert C. Martin.

Clean Architecture is a software architecture pattern that emphasizes separation of concerns whilst promoting a modular, testable, maintainable and loosely coupled codebase. In this architecture, core business and application logic are independent of the UI, database, and external infrastructure services [69].

The Clean Architecture pattern is typically represented by four concentric circle layers where each layer has unique responsabilities:

- **Entities**: represent the high level business rules of an application and is the layer that

is less impacted by changes on external layers, it typically comprises database entity classes (or data access objects (DAO)), repository interfaces and external services interfaces;

- **Use Cases**: entail the application specific rules and how data flow is handled to and from the entities layer. This layer holds the various features that an application provides and how to interact in the databases, as well as provides the various interface contracts or data transfer objects (DTO) that are used on the interface adapters layer;

- **Interface Adapters**: define the entry point of the application. This layer is where the UI, API endpoints and external service providers normally reside;

- **Frameworks and Drivers**: defines the implementations of database providers, external services and infrastructure.

These layers have adopted different names overtime and throughout the developed work we will use a different notation for the previous layers respectively: **Domain/Core, Application, Presentation/Web and Infrastructure**. The primary principle of this software architecture pattern is the **Dependency Rule** which states that inner layers should not know anything about the implementation details of outer layers. This principle is also correlated to the **dependency inversion principle (DIP)** [4] which aims to provide interfaces on the inner layers instead of implementations. By defining interface contracts at the inner layers, the application becomes flexible to change, allowing outer layers to be easily swapped or modified without requiring additional modifications on the inner layers. Following this approach paves the way for building a system that is maintainable, testable and extendable. However, due to having to ensure that the various inner layers aren't dependent of their outer ones, this poses a initial learning curve for developers to build applications following this architectural pattern, slowing the development process.

On the other hand, the **N layer architectural pattern** is easier to grasp and to develop applications but due to its high coupling between layers it can become harder to maintain, scale and test compared to the Clean Architecture [17] as seen in Figure 2.20. In addition to the software architecture pattern, some design patterns were also employed on the developed work, specifically **the repository and unit of work, mediator and command query segregation patterns**.

The **repository pattern** serves as an abstraction for interacting with databases and is also typically integrated with the **unit of work pattern** to provide transaction capabilities on the backend system [5].

[4] System Design: Dependency Inversion Principle - https://www.baeldung.com/cs/dip

[5] Implementing the Repository and Unit of Work Patterns in an ASP.NET MVC Application -https://learn.microsoft.com/en-us/aspnet/mvc/overview/older-versions/getting-started-with-ef-5-using-mvc-4/implementing-the-repository-and-unit-of-work-patterns-in-an-asp-net-mvc-application

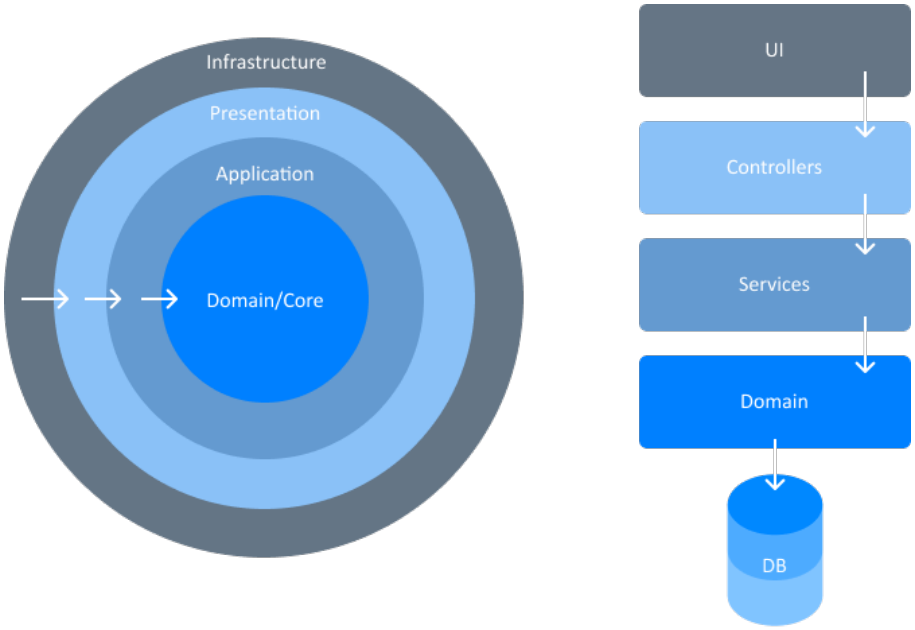


Figure 2.20: Clean Architecture and N Layer Architecture - Software Architectural Patterns

Lastly, **the mediator pattern** is a behavioral design pattern that enables decoupling of objects by introducing an mediator object between two objects that facilitates their communication, instead of direct communication between each other. By using this pattern we can create message contracts for communicating with the target objects while also decoupling them since we use the mediator as the means of communication between each object [6].

To cease the backend development chapter, a decision must also be made on how the data will be stored, including typical medical record data, as well as the sensors recordings and also how the backend API will be hosted. IoT devices are expected to reach 73.1 zettabytes (which is over 73 trillion gigabytes) of data generation by the year 2025 and it has been estimated that over 10 billion active IoT devices were being used in 2021 [31]. This presses the need of efficient computing models and storage solutions that are capable of withstanding these big volumes of data while also being economically viable.

Cloud computing is robust due to the panoply of provided services [77], the resilience due to replication and horizontal scaling provided out of the box and their ease of integration both of different types of databases and programming languages. The use of cloud structures is becoming more adopted due to the fact that traditional dedicated servers in the event of a server fault, all the sensor data that was being transmitted becomes lost during that failure window. In the case of cloud computing models, if for example, a replication service is enabled and if only that server fails and not the whole datacenter, a new one is provisioned after the fault is detected and the overall downtime is shortened, issuing a quicker recovery and reducing the overall sensor data

[6]Mediator design pattern - https://www.geeksforgeeks.org/mediator-design-pattern/

loss. Additionally maintaining and buying dedicated servers to manage IoT data can be an expensive option depending on the resources that are needed, in the case of public clouds, the services are way more economically accessible and maintenance of such equipment is not of our concern. Cloud databases also provide a range database paradigms, let it be Relational, Non-Relational (or document based), Graph Oriented or Time-series, where traditional servers need manual configuration and setup of these instances.

Despite mentioning the various advantages of using a cloud computing model to host the backend application, due to time constraints the developed work will only be tested locally as it is already quite complex due to having two frontend layers, a sensor layer and a backend layer.

The only cloud service that will be used in the proposed work will be **Azure Blob Storage** which is optimal for storing blob files (or binary files). In the proposed work blob storage will be used to store media content related to physiotherapy exercises, patient and physiotherapist profile images and clinic images.

For storing the system data there were two options, **relational databases or document-based databases**. **Relational databases** like (SQL Server, MySQL, PostgreSQL) are best suited for database designs that require well defined entity relationships, data accuracy and also when the subjacent applications require that their database must have ACID properties which can be critical for some systems [54]. Another approach could be to adopt a document-based database which is denormalized and that leverages BASE properties which are optimized for cloud usage like the case of MongoDB, Azure Cosmos DB, Amazon Dynamo DB [81]. Since the developed work database design uses many table relationships and since the cloud computing model was discarded from the backend application, the backend will use a relational database, specifically PostgreSQL for storing the systems data. The choice for selecting PostgreSQL is due to being open-source and also being widely adopted in the tech industry, but another important factor is due to its support for **set returning functions** [7], which enables PostgreSQL to generate series of values of a given type using start and stop thresholds and also a step value for generating said series. This feature will be used later on the developed work for dynamic generation of appointment availabilities from physiotherapists without having to store these in the database.

2.4.4 Real-time protocols and notification systems

When conceiving a remote health care system that supports chat capabilities and push notifications, real-time protocols should be explored and analyzed, as well as study the intricacies of a push notification system that has to work on both Android and also on the web.

There are many real-time protocols and implementations to choose from but on this thesis we will only go through how **long polling, websockets, server-sent events** real-time

[7] Set Returning Functions - https://www.postgresql.org/docs/current/functions-srf.html

communication strategies work and also deep dive into **.NET's SignalR library** internals and explore how it's able to provide real-time communication in web applications.

2.4.4.1 Notification systems

Notification systems are an important factor when building interactive web and mobile applications. These systems ensure that users stay informed about time sensitive events, incoming messages, and actions that require attention. They also promote engagement with the application, as these notifications are popped into the user's screen even when they are not directly interacting with the application. This is possible due to notifications being able to operate in both foreground and background scenarios.

For implementing notifications in the *KNEEMOR* system, we must explore how **Android notifications** work, and also explore the **Notifications Web API from Javascript**.

Android notifications are message containers that get displayed on the user's mobile screen which provide mobile app reminders, present time sensitive information and allow users to take necessary actions like answering a phone call or cancelling some ongoing service that is being displayed as a notification [48]. Notifications are managed through Android's **NotificationManager service**, which acts as a middleware between a application and the Android operating system's notification system. Starting from Android 8.0 (API level 26), all notifications must be assigned a channel [48]. Additionally, for Android 13 (API level 33) and onwards, a permission must be added into the AndroidManifest to support notifications, specifically **`POST_NOTIFICATIONS`** permission.

A **channel** defines user preferences for a given notification type. An Android app might have different channels for different functionalities. For example, in the *KNEEMOR* system, there could be a chat channel for message notifications and a separate channel for handling the user preferences in terms of reminders and alerts. When defining a channel, you must specify a **channel ID** which uniquely identifies the channel, a channel **name** which serves as a title in the notification settings, and also the level of **importance** which defines how underlying channel notifications behave in terms of interrupting the user visually and audibly, the higher the importance the more disruptive the notification is. After creating the notification channel, we can publish notifications through it. The structure of a typical notification includes **the channel ID, the notification ID, a title, text, a small icon, a large icon, the application name and zero or more notification action buttons, and the notification's tap action**. By default, when tapping a notification, the current app activity is opened to the user, if it isn't already open. We can change this behavior by assigning other activities or functionalities that get triggered upon tapping a notification or through one of its available action buttons.

Since the developed work uses React Native to build the Android application, a library had to be chosen for providing support for notifications. Initially react-native-push-notification library was used, however during development it was detected that the library only had support for older Android versions and required code updates to be

compliant to the newer API changes. Additionally, the library itself was also no longer being maintained. Due to these reasons, a newer library, **Notifee**[8], was adopted to achieve the same goal.

In the physiotherapist application, and since there's access to progressive web app capabilities, meaning there's also access to service workers, we can leverage the **Notification API**[9] to deliver push notifications. To send notifications through the Notification API, a permission must be first requested to the user. Once the permission is granted, we can then create notifications by defining **a title, text, image and respective event handlers that perform actions like navigation around our application**. Despite progressive web apps not being widely supported yet across all platforms, and the Notifications API also having limited browser and device support, it was still the best option for providing notifications to the physiotherapists application without consuming excessive development time.

2.4.4.2 Real-time communication strategies

Long polling is a unidirectional real-time communication strategy where a client application periodically send requests to poll data from the server [64]. These periodic requests for polling data simulate real-time communication, but in reality, this strategy works in intervals of time instead of real-time. The communication process begins with a client issuing a request to the server and waiting for a response. If new data is available on the server, then the server responds to the client and the connection is terminated. If no data is available at the time and the timeout window is reached, then a timeout response is issued to the client, prompting it to issue a new request to poll data as depicted in Figure 2.21 (a). This process continues and is repeated periodically making it feel like there is real-time communication. The strategy is unidirectional since the server cannot push data to the client unless the client first creates a request. While this strategy is widely adopted and easy to implement, it is resource consuming since for polling new data a new request must be always created.

Server-sent events on the other hand, maintain a long-lived HTTP connection that allows the server to push data proactively to the client as it becomes available. The communication process begins with the client establishing a HTTP connection with the server. Once the connection is established, the server is able to push messages or events to the client, which awaits and handles them until either the server or the client closes the connection as illustrated in Figure 2.21 (b). It is also a unidirectional process, since the client can only initiate the connection to the server and then can only receive incoming events pushed by the server, being unable to produce new requests over the same connection[64].

Lastly **Websockets**, are a bidirectional (or full-duplex) real-time communication strategy which rely on a single persistent TCP connection for allowing data exchange between both client and server. The process begins with the client performing the websocket

[8]Notifee - A feature rich notifications library for React Native - https://notifee.app/
[9]Notifications API - https://developer.mozilla.org/en-US/docs/Web/API/Notifications_API

handshake which basically passes a HTTP Upgrade header to server, so it can know that it can proceed establishing a websocket connection. After the handshake is successful, the connection between server and client is established and both parties can start exchanging data in both directions as seen in Figure 2.21 (c). Websockets provide low latency, reliable channels for exchanging data and is a great choice for providing real-time communication for applications. In the developed work, this will be the real-time communication strategy that will be used.

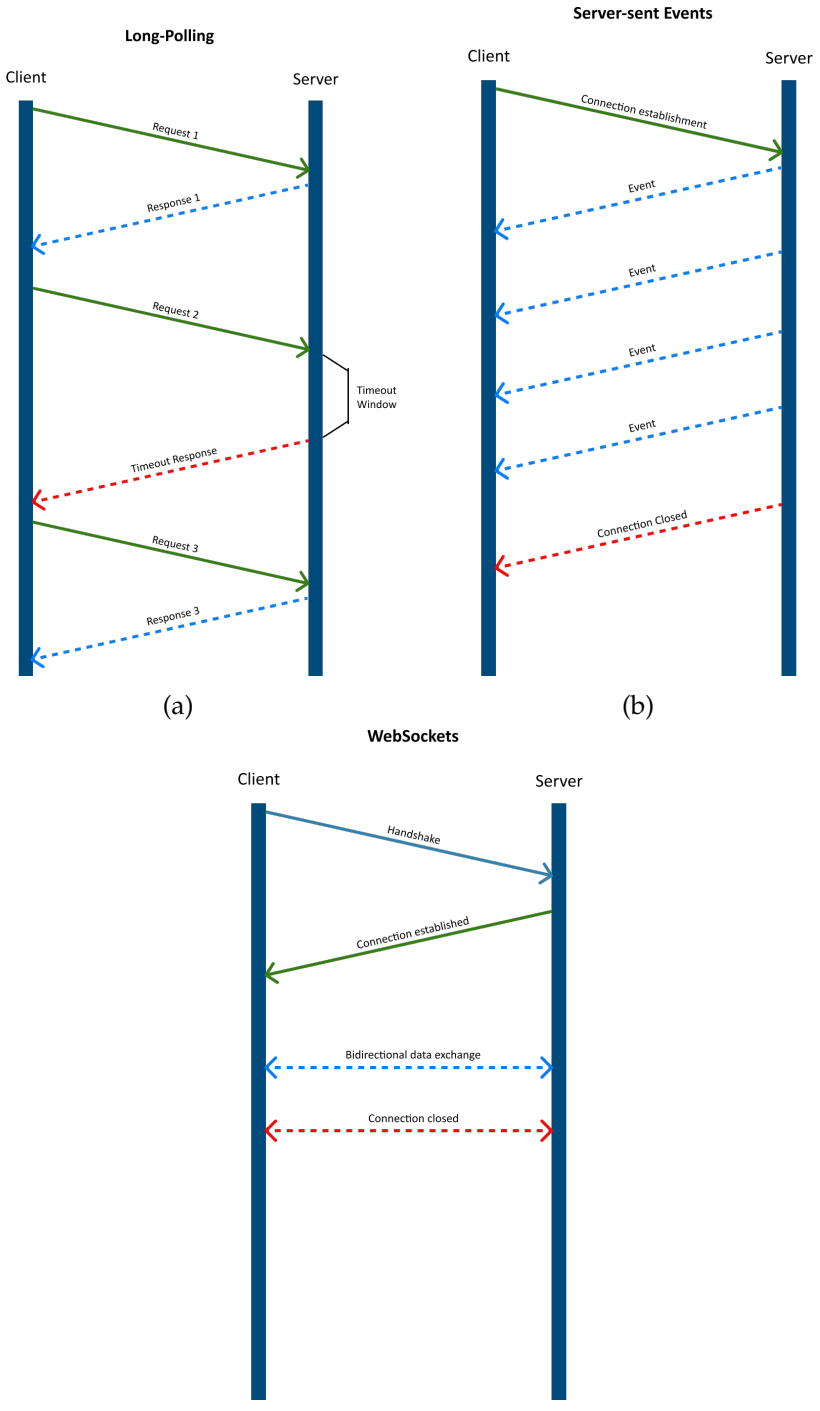


Figure 2.21: Different strategies for providing real-time communication. (a) Long-Polling, (b) Server-Sent Events, and (c) Websockets.

2.4.4.3 Real-time communication using SignalR

Having gone through the different real-time communication strategies, we now address the library used in the developed work to provide the chat functionality for *KNEEMOR* system, which is **.NET's SignalR library**.

SignalR is a open-source library developed by Microsoft which enables real-time communication capabilities for web applications. It allows server-side code to push content to connected clients instantly without requiring the client to poll or request data [32]. This is achieved through an API that SignalR provides for creating server-to-client remote procedure calls (RPC). Through these RPCs, the server can invoke functions on connected clients, and clients can invoke functions on the server as seen in Figure 2.22. This API is defined in what is called a **SignalR Hub**, which holds the various RPC methods that can be used by the clients. The communication process begins with a client connecting to a SignalR Hub endpoint. A transport negotiation then takes place, where the client and server agree on the protocol to use for establishing the Hub connection. Once the transport negotiation ends, the SignalR connection becomes established and both parties can start invoking RPC calls. In case of disconnects or network faults, SignalR is also capable of handling automatic reconnects to hubs.

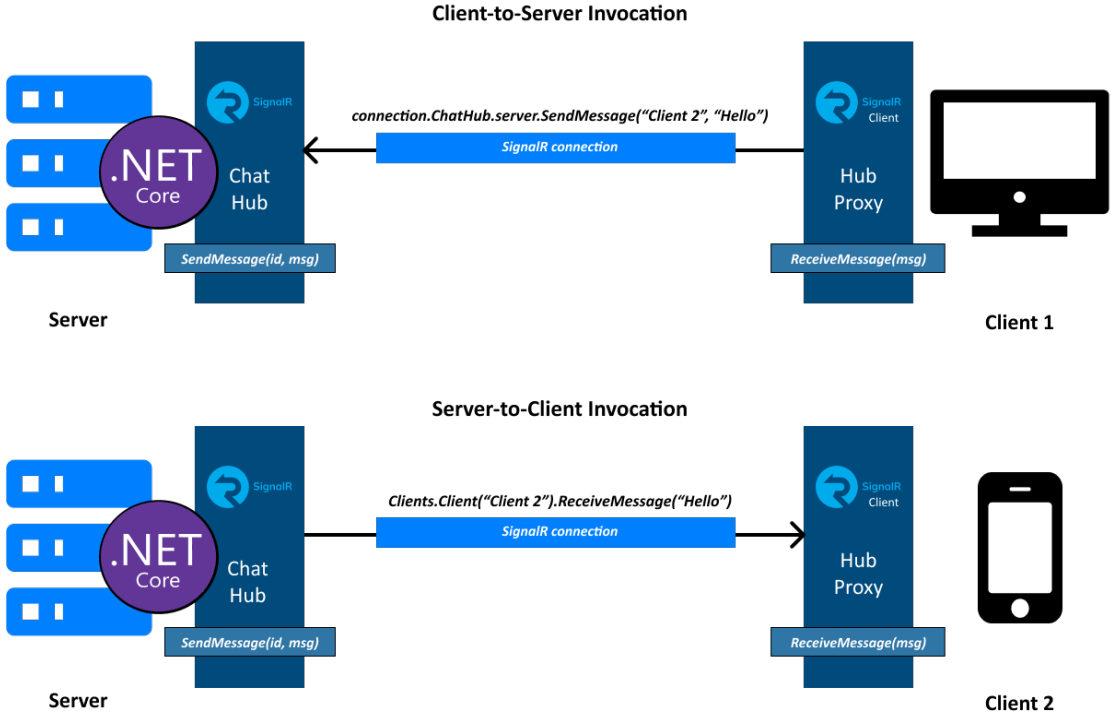


Figure 2.22: SignalR RPC Invocations Detailed - adapted from Microsoft Documentation [32]

Also, when performing communication with SignalR, we can opt by sending messages to only one specific client or to multiple clients using **groups**, this is advantageous when there is need for broadcasting information to various clients simultaneously. SignalR also provides secure communication by implementing .NET Core's authentication and

authorization systems [12], ensuring that only authenticated and authorized clients can connect to the SignalR hub endpoint. Additionally SignalR library has a wide support for different clients and devices, making it an ideal choice for being used both in the physiotherapist and patient applications.

In the developed work, SignalR is used in conjunction with the previously mentioned notification systems, to provide chat functionality and daily reminders to connected clients, this makes both applications more interactive.

2.4.5 Wearable Motion-Sensor Infrastructure

When prototyping a wearable motion sensor, there are various hardware factors to take into account like: which development boards to use, which sensors to adopt for achieving the required measurements, which actuators to integrate to provide interesting biofeedback to patients and also what communication protocols should be used to transfer the relevant sensor data to the external applications. In this section, a brief comparison is made between existing hardware options and also it is established the hardware ecosystem that will be used in this thesis.

2.4.5.1 Development boards

A development board is piece of hardware that helps in the prototyping and development of embedded systems. Development boards are composed a microcontroller chip that holds a central processing unit (CPU), memory, digital and analog I/O pins, different types of communication interfaces, various connectors, timers, interrupts and can also provide onboard integrated actuators depending on the model of the development board [25]. The digital and analog I/O pins are tied to analog-to-digital converter (ADC) and digital-to-analog converter (DAC) where the digital-to-analog converters transform binary values to analog values where the latter does the inverse. The overall management of resources and control flow is assured by the microcontrollers CPU and the memory of the microcontroller provides volatile and non volatile storage. When developing an embedded application, the application code is uploaded to the flash memory (non volatile) of development board [25] and when the power is cut off from the development board, the code that was previously uploaded isn't lost. Instead the application code remains persisted in memory so that when the development board is turned on again, it can operate accordingly to the previously uploaded code. The development boards provide an easy way to develop embedded applications due to their built-in programming interfaces, accessible connectors like USB, onboard capabilities like communication protocols (WiFi, Bluetooth) and even onboard actuators. There are various choices to pick from ranging from Arduino boards, Raspberry Pis, Adafruit boards and ESP32 development boards [4]. From the ESP32 family boards, the model ESP32 WROOM 32D DevKitC provides onboard WiFi and Bluetooth modules, 4MB of flash memory and is supported on Arduino IDE

development kit opening a wide range of libraries to employ for application development [23].

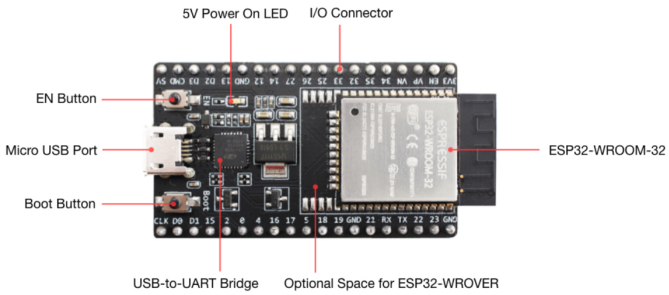


Figure 2.23: ESP32 WROOM 32D DevKitC[10]

Due to also being a relatively cheap option compared to like a Raspberry Pi board or other options, it makes an ideal candidate to develop embedded system prototypes at a low cost, hence this was the board that was picked for developing the sensor prototype.

2.4.5.2 Sensors

Sensors are the components that allow development boards to be able to perceive and measure changes around its surrounding environment. Through sensors it is possible to measure movement, temperature and other types of indicators.

When building a wearable motion-sensor prototype, the selection of the sensors is crucial, since picking the wrong sensors can impact the end result of the prototype. And since we are dealing with measurements over the knee joint angle, it makes sense to evaluate the following types of sensors: **flex sensors and inertial sensors.**

A **potentiometer** is also another worthy mention, despite not being classified as a sensor, as it also allows to detect mechanical movement hence it should be also addressed.

It is composed of a resistive strip, a wiper and three terminals to allow the measurement. The resistive strip is what provides the electrical flow to go through the circuit and the wiper controls the applied resistance to the electrical current affecting the output voltage [2]. If the wiper is slided fully to the positive terminal, the output voltage is maxed, if the wiper is slided towards the negative terminal the output voltage decreases.

There are several types of potentiometers that allow for tracking movement, the most common ones used for rotating mechanical rigs are **rotary potentiometers** which measure rotational movement by regulating a knob. As for tracking linear movement we have **linear potentiometers** that use wipers that slide along the resistive strip and depending on its displacement it is possible to infer movement.

String potentiometers (or stringpots) can also measure linear movement but work differently from a traditional linear potentiometer. These potentiometers are composed of

[10]ESP32 WROOM 32D DevKitC Development Board - image from - https://www.botnroll.com/pt/esp32/4395-esp32-devkitc-32e-placa-de-desenvolvimento-espressif.html

(a) (b)

Figure 2.24: Different types of potentiometers. (a) Rotary potentiometer. (b) String potentiomter (or stringpot). [2]

a flexible measuring cable, a spool, a spring and a rotational sensor (which typically is a rotary potentiometer but can also be an encoder) as depicted in Figure 2.25.

The flexible cable is attached to the object we want to perceive the displacement while the stringpot remains stationary on another end.

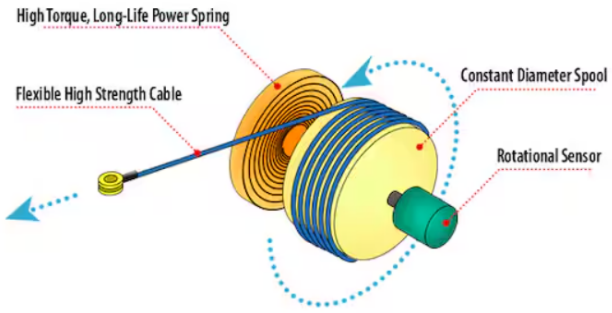


Figure 2.25: String potentiometer composition. [28]

As the cable is extended by movement, it causes rotation of the spool and sensors shafts which in turn creates a proportional electrical signal to the displacement of the movement. The produced electrical signal is then converted to a linear output which describes the linear displacement of what we are tracking.

Flex sensors or bend sensors are also variable resistors that measure the electrical resistance when bending or flexing the sensor. It is composed of a flexible substrate, a segmented conductor, conductive ink and two pins for retrieving the measurement data. The electrical resistance is at its lowest when the sensor is completely stretched and the resistance increases gradually as the sensor is bent [76]. These types of sensors can only be bent in one direction otherwise they incur in permanent damage and cease working correctly as seen in Figure 2.26.

[11] Flex sensor - Image from - https://learn.sparkfun.com/tutorials/flex-sensor-hookup-guide/all

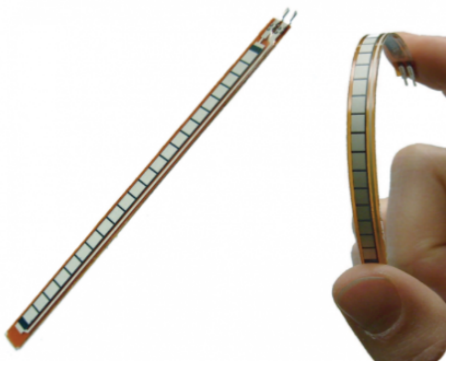


Figure 2.26: Flex sensor and bending direction.[11]

During early development of the thesis, there was some experimentation with this type of sensor but due to its maximum angle limitation it ended up being discarded since the knee joint angle can reach up to 140º degrees in maximum flexion as seen in the research hence making it an unviable choice for measuring the full range of motion of the knee joint. Also another aspect to these sensors is that they typically require textile fabrics to hold them in place and due to this fact they are more cumbersome to wear and more constricting on the knee joint movement [36].

Inertial sensors or inertial measurement units as previously discussed are sensors that hold 6-DOF in case of only comprising an accelerometer and a gyroscope, or can they can be sensors that also comprise a magnetometer which adds up to 9-DOF. They are small in size, compact and offer reliable readings through the use of sensor fusion algorithms that are typically processed at the microcontrollers end. In previous works, the MPU-9250 IMUs[12] were used to perform the respective joint angle measurements, in this thesis we explore its successor the ICM-20948 [13].

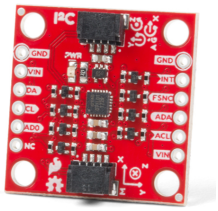


Figure 2.27: ICM-20948 IMU[13]

It terms of functionalities, both sensors don't differ much in their sensor value ranges as seen in Figure 2.28 and also offer the same SPI and I2C communication interfaces. However, there's a slight improvement in the magnetometer value range and also the DMP data fusion algorithms from the ICM-20948 use 9-axis fusion while its predecessor uses 6-axis fusion.

[12]MPU-9250 IMU - https://invensense.tdk.com/products/motion-tracking/9-axis/mpu-9250/

[13]SparkFun 9DoF IMU Breakout - ICM-20948 with Qwiic connector - Image from - https://www.sparkfun.com/products/15335

Feature \ Part		MPU-9250/55	ICM-20948	ICM-20948 Compatibility
Accel Range		±2g to ±16g	±2g to ±16g	Equivalent
Gyro Range		±250 to ±2000 dps	±250 to ±2000 dps	Equivalent
Magnetometer Range		4800µT	4900µT	Superset
DMP		Yes with 6-axis fusion	Yes with 9-axis fusion	Superset
Communication interfaces	I2C Slave	400Khz	400Khz	Equivalent
	I2C Master	Auxiliary, 400KHz	Auxiliary, 400KHz	Equivalent
	SPI Slave	1Mhz 20Mhz sensor data read	7Mhz	Compatible
I2C Address		7b'110100x	7b'110100x	Compatible

Figure 2.28: Functionality comparison between MPU-9250 and ICM-20948[14]

These TDK Invensense IMUs offer embedded proprietary data fusion algorithms whose processing can be offloaded to the IMUs themselves instead of being processed at the development boards microcontroller. These fusion algorithms are applied through the digital motion processor (DMP) that is embedded in the IMUs. The advantage of using DMP for data fusion is that it removes some processing overhead on the microcontroller enabling it to use its resources for other tasks, provides some degree of self calibration through the IMUs computation power and offers multiple data representations ranging from raw values, rotation vectors and quaternions.

In the developed work, the SparkFun 9DoF IMU Breakout - ICM-20948 with Qwiic [13] connector was selected due to providing a stable library to work with, and due to its interface connector. Qwiic interface connector[15] requires no soldering, is daisy chainable meaning it can connect with other Qwiic interfaces and share the same I2C bus for transmitting and receiving data and is easier for prototyping due to the connector aggregating all for required pins for I2C communication which will be the communication interface that will be used in this thesis.

2.4.5.3 Feedback actuators

Feedback actuators are crucial for providing feedback in embedded systems. They are responsible for receiving input signals made by external stimuli, normally by sensor data and translating that data into relevant feedback form let it audible or visual. These devices help drive engagement between the sensor prototypes and their wearers and also provides useful indicators of how the prototype is operating.

We will go through three types of feedback actuators: **LEDs, assembled LEDs and buzzers**.

A light emmiting diode (LED) receives electrical current that is transformed into light. Their composition consists of two pins, a anode which is the positive terminal of the led where electrical current enters and the anode which is the negative terminal and by where

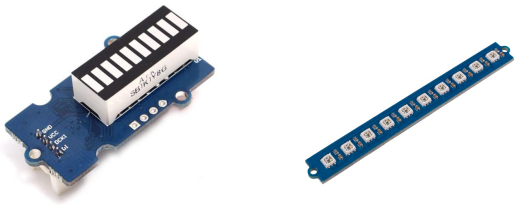
[14]Migrating from MPU-9250 to ICM-20948 - https://invensense.tdk.com/wp-content/uploads/2018/10/AN-000146-v2.0-TDK_Migration_MPU_9250toICM-20948.pdf

[15]Qwiic Connector Explained - https://www.sparkfun.com/qwiic

the current exits, while RGB LEDs may have an additional pin to control the color emitted by the led [2].

Assembled LEDs are pre-built programmable LED components that offer various integrated programmable LED lights that can be easily manipulated through accessing their registers. These components transmit more information due to their visual composition and animation possibility.

In previous works, the wearable motion-sensing prototypes didn't support any form of visual feedback integrated in them only through their mobile application counterparts. As an improvement on past works it was decided to include an assembled led that shows progress on the movement that is being made, more precisely the angle that is being read from the sensors. For the developed work there were two hardware options for implementing this feature, either the **Seeed Grove - RGB LED Stick or the LED Bar** as depicted in Figure 2.29.



(a) (b)

Figure 2.29: Assembled programmable LEDs from Seeed. (a) Grove LED Bar.[15] (b) Grove LED Stick.[15]

The Grove - LED Bar was selected due to its more compact size and for providing 10 programmable LEDs (8 green, 1 yellow and 1 red) which can provide a gradient effect when lighting up the various LEDs. It provides a grove connector that is easy to prototype with and also to program the various LEDs there is available an Arduino library for its purpose. This component was enough for the visual feedback requirement since we need to only establish the maximum and minimum angle during each exercise, and with the available range of LEDs we can translate these parameters into a LED lighting state, being that at initial position all leds are off and as the knee bends to the maximum angle the gradient of colors are lit up.

Lastly **buzzers** are feedback actuators that produce sound based on the current that flows through them. On the thesis it was adopted an electromagnetic buzzer to provide audible feedback when an exercise has been completed.

[15]Seeed Grove LED Bar - Image from: https://wiki.seeedstudio.com/Grove-LED_Bar/

[15]Seeed Grove LED Bar - Image from: https://wiki.seeedstudio.com/Grove-RGB_LED_Stick-10-WS2813_Mini/

2.4.5.4 Network Protocols

When developing a wearable sensor prototype it is crucial to evaluate which network protocol to employ in order to deliver the sensor data to the target application or device that will visualize this data.

Network protocols are standardized rules and conventions that define how data is transmitted, received, formatted, and interpreted, ensuring seamless communication between devices regardless of their underlying hardware or software.

There are several network protocols used in IoT environments including: **BLE, WiFi, Zigbee, ZWave, LoRaWAN and others**. In this section only two options were discussed, since these two were the potential options to be considered for the thesis: **WiFi and Bluetooth Low Energy**. In this section the strengths and weaknesses of each protocol are discussed, as well as the reason for a given protocol being chosen for the developed work.

When deciding which network protocol to use, 8 core parameters should be taken into account regarding network protocols: **data rate, range, power consumption, interoperability, scalability, cost, topology and security**[34].

Based on the wearable sensor prototype specifications, the main focus should be on following parameters **data rate, range, power consumption and interoperability**. The remaining parameters are not of particular interest due to the small scope of this prototype and since it will only communicate with one device node which will be a patient's mobile phone. The security parameter can also be discarded, since the sensor data doesn't uniquely identify any patient and the data being only useful in the underlying patient application.

Data rate consists on the amount of information that is transmitted in a given time period. It follows a unit of measurement of bps (bits per second), and nowadays the most common orders of magnitude for this unit of measurement are Mbps (Megabits per second) and Gbps (Gigabits per second). In this parameter, WiFi has the advantage since it can support high data rate transmission in comparison to BLE which is capped at 2Mbps on BLE 4.2 and BLE 5 [21].

The **range** parameter relates to the maximum executable distance for interchanging data between two nodes. WiFi here also has the advantage due to its wider reach compared to BLE that has significantly shorter ranges.

The **power consumption** which is related to a node's energy consumption and battery life autonomy, BLE here has the advantage over WiFi, as it consumes significantly less energy compared to WiFi making it ideal for battery-life constrained devices.

Lastly in terms of **interoperability**, both WiFi and BLE are widely adopted from a wide range of mobile devices nowadays. WiFi has always been more prevalent while BLE started being adopted in the recent years. BLE and WiFi are agnostic of operating systems and work both on Android and iOS. The choice here depends on how much device support the prototype should provide. If the aim is to cover a wider range of device support, then WiFi would be the perfect option but since BLE is becoming more

prevalent on mobile devices as of today, it might be a better option to choose BLE due to its power consumption being reduced and due to needing to be close to the wearable sensor prototype.

Based on these parameters, the best option for the wearable sensor prototype as network protocol is BLE due to its low energy consumption, wide adoption and interoperability. To give the reader a better understanding of how BLE works, a brief introduction will now be given.

BLE is a wireless low-power network protocol widely used in various applications and also in IoT environments to provide uni-directional or bi-directional communication between two Bluetooth devices. The protocol is comprised of three main layers: **controller, host and application layer** as depicted in Figure 2.30.

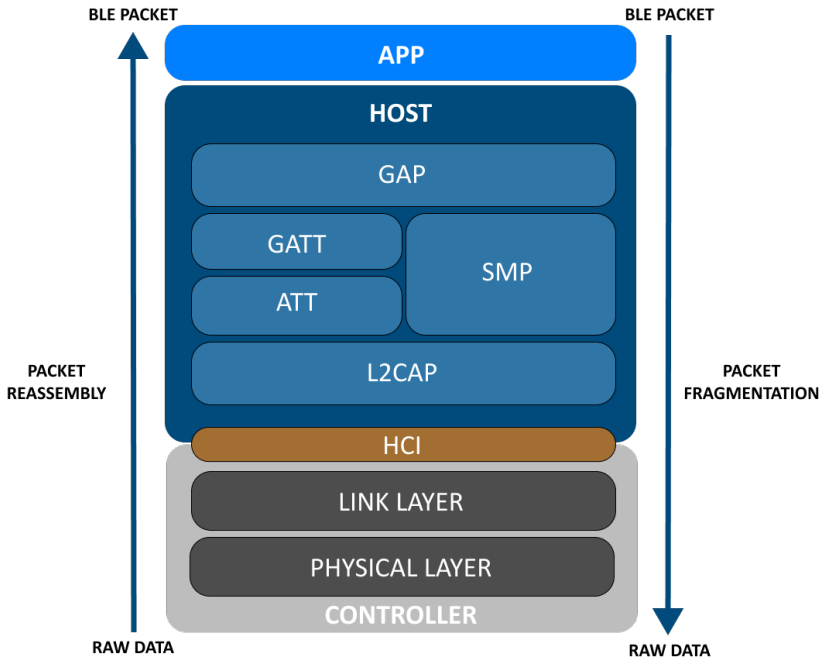


Figure 2.30: BLE Protocol Stack - Illustration adapted from *"Performance Evaluation of Bluetooth Low Energy:A Systematic Review"* [70]

The **controller layer** is defined by the Physical Layer and Link Layer, and its main responsibility is to provide an interface named Host Controller Interface (HCI), which bridges the communication between the controller layer and host layer [21].

The host layer houses four components: **Logical Link Control and Adaptation Protocol (L2CAP)**, **Attribute Protocol (ATT)**, **Generic Attribute Protocol (GATT)**, **Security Manager Protocol (SMP)** and **Generic Access Protocol (GAP)**.

The **Logical Link Control and Adaptation Protocol (L2CAP)** is responsible for performing packet fragmentation and reassembly for between layers. Inbound raw data is encapsulated into a standardized BLE packet format that follows the ATT and SMP specifications, while an outbond BLE Packet is retranslated into raw data format so it can be relayed to the lower layers.

The **Security Manager Protocol (SMP)** is responsible for the device pairing and bonding functionalities, security algorithms responsible for data encryption/decryption and also manages authentication of the BLE devices. The protocol defines upon formation of a BLE link what device has the role of master or slave, where the master is the device that initiates the link establishment between the other device that is the slave (which is advertising himself in the network).

The **Attribute Protocol (ATT)** defines the communication roles between two devices, if they assume a server role or a client role (or both) and is additionally responsible for organizing data into **attributes**, which are composed of a handle, a Universally Unique Identifier (UUID), a set of permissions and a specific value [70]. The **Generic Attribute Protocol (GATT)** is responsible for the data management and profile exchange between the GATT server and GATT client and also encapsulates the ATT protocol in its structure. A profile is a definition of behaviors and metadata that describe the type of data a Bluetooth module transmits.

The data itself follows an hierarchical structure which aggregates a set of components named **services**, which hold a number of **characteristics** which group and contain data as seen in Figure 2.31. A **service** contains a collection of **characteristics** each with its own specific type. Each **characteristic** holds a descriptor which is metadata information about the characteristic, a value and a set of properties. The properties of a characteristic define what type of operations the GATT client can perform, the available operations are the following: **broadcast, readable, writable and notifiable**. Broadcast is used for advertising packets and discovery of BLE devices, readable and writable properties define if the characteristic can be read or written to by the client and lastly the notifiable property defines that the characteristic may be updated by the server and the client can receive the state update accordingly and read said value.

The **Generic Access Protocol (GAP)** manages the discovery process and connection establishment of BLE devices as well as assign device roles to BLE devices. The GAP defines four roles for BLE devices, these roles are: **broadcaster, observer, peripheral and central**. These roles are associated to two communication modes defined in the GAP, **broadcasting and connection modes**.

Broadcasting mode uses broadcaster and observer roles, where one of the devices assumes the broadcaster role of advertising BLE packets to any device in the vicinity and the other device behaves as the observer role where it keeps scanning in periodic intervals for available advertised packets. The communication mode does not require a BLE link to be fully established and is the fastest way to transmit data to more than one device.

The connection mode instead relies on pre-established connection that must be performed before transmitting any data. In this connection mode, the **central role** device scans the network for advertising packets and is the party that initiates the BLE link connection. The peripheral role on the other hand, is the device that advertises packets and accepts connections that are triggered by the central. The central role defines the settings and tendency of packet exchange by which the peripheral role should follow.

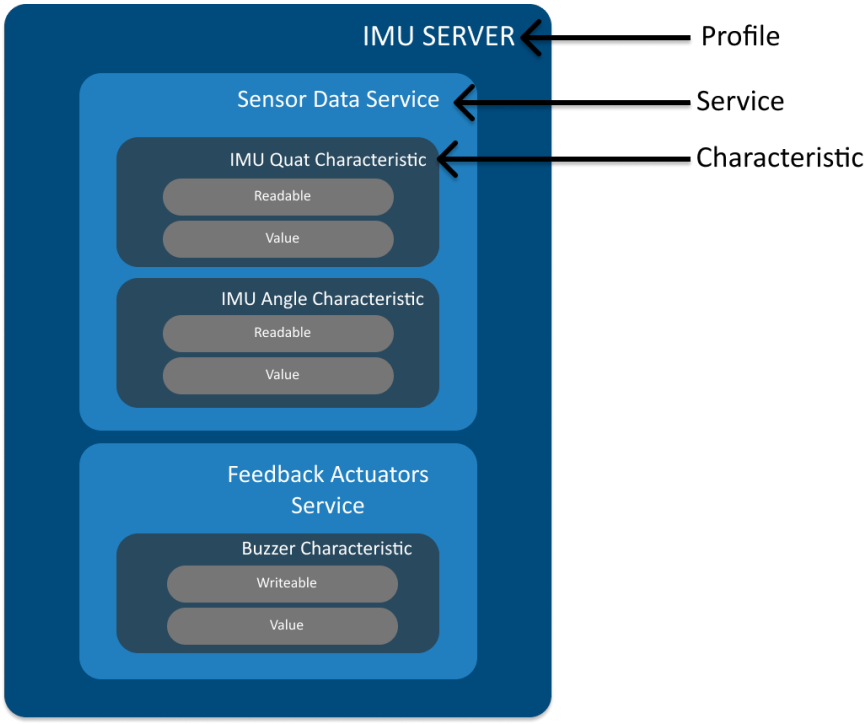


Figure 2.31: BLE GATT Hierarchical Structure

In the proposed work, the Wearable Sensor Prototype will act as the peripheral and the patient's application will act as the central. The communication will be bi-directional since despite the wearable sensor prototype accepting connections and broadcasting afterwards the sensor data, it is required that the patient's application commands the timing of when to start collecting sensor data as well as the time to stop the collection.

2.5 Commercial Solutions

There are several existing commercial solutions that provide e-rehabilitation features while also driving user engagement to do exercise programs. Some worthy mentions are the following:

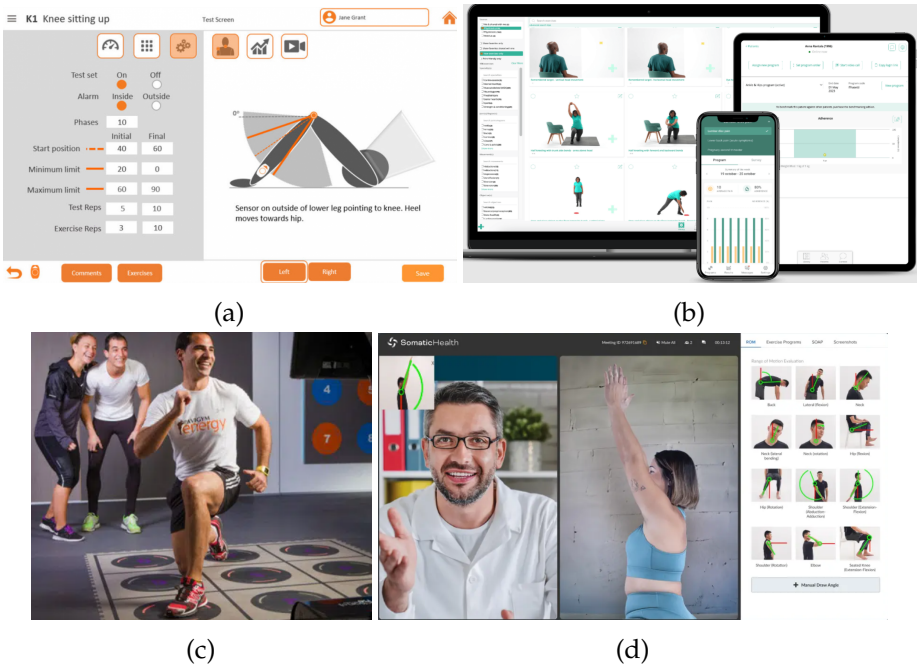
- **Exergame** - is an ecosystem of fitness solutions that drives people's engagement in doing exercises through use of gamified infrastructures, hence the name Exergame (exercise + gaming) [24]. Despite most of the products being more fitness oriented, Exergame also provides with solutions that are applicable to the area of rehabilitation and physical therapy. The solutions tWall, Makoto, Square 3.0 (depicted in Figure 2.32c), Dynavision help improve reaction times, coordination, motor mobility and even sensory skills while also providing tracking capabilities to show scores and progress of the people's evolution throughout the exercises. The gamified aspect of these solutions makes doing exercise not a task but leisure which itself is beneficial and motivating for patients.

- **BPMpathway** - is a remote rehabilitation system which uses a mobile application and sensor on the ankle which tracks knee joint health [15]. It provides range of motion tracking capabilities, historical data of the progression of the knee range of motion, two-way communication between patient and physiotherapist, exercise assignment and parameterization and also provides patients with visual feedback through the display of an avatar with exercise that is being performed as seen in Figure 2.32a. Some features of this system inspired the conceptualization of some functionalities on the Frontend that is going to be implemented in this thesis.

- **GPEP** - is a mHealth solution for muscle and joint rehabilitation, it provides information about exercises and videos on how to perform them, offers exercise scheduling, progress monitoring, allows to share progress with other users and also provides the ability to attend live rehab sessions. The strong point of this application would be the live rehab sessions and the information it provides on each exercise program.

- **Physitrack** - is a remote physiotherapy system based on a web and mobile application. It allows health professionals to prescribe exercise programs through a wide library of exercises and assign them to the patients [55]. These exercises are comprised by videos which can be followed by their patients on a mobile app (PhysiApp) as seen in Figure 2.32b. The patients, when first interacting with the application, have a small survey that provides triage, and in critical cases the system forwards the user to book an appointment, either remotely or presential. Additionally, health professionals can also add new images and videos of exercises in case they want to extend the system or replace with their own videos.

- **SomaticHealth** - SomaticHealth (or previously known as BlueJayHealth) is another remote physiotherapy platform similar to Physitrack that provides exercise program customization and assignment, patient progress tracking and appointment scheduling [63]. The particularity of this system is that it infers range of motion from body joints through an AI-powered digital goniometer and through video capture as depicted in Figure 2.32d. On each exercise session, the exercise is recorded and the results from the exercise performance can be later on extracted as a video or through a pdf report. The platform also supports real-time video calls while performing exercises.



(a) (b)

(c) (d)

Figure 2.32: Commercial solutions with applicability on physiotherapy. (a) BPMpathway. (b) Physitrack. (c) Exergame - PaviGym Square 3.0. (d) SomaticTelehealth.

2.6 Conclusions

Telehealth and e-rehabilitation are medical services that are provided using technology to connect patients and healthcare providers remotely. Remote health allows patients to access healthcare services from their homes or wherever their location might be without having to visit a medical facility. Correlated to this, medical practices that require physical intervention can also benefit from the e-rehabilitation services like the case of physiotherapy.

However, implementing an e-rehabilitation has its challenges, including ensuring the privacy and security of patient data, guaranteeing the reliability and accuracy of the sensor readings, and also ensuring that the technology is accessible for both patients regardless of their disabilities and also for physiotherapists in terms of being economically viable and accurate, to introduce these systems in their practice.

Apart from studying the overall knee anatomy, it was also explored the main advantages and disadvantages between optical and non-optical motion tracking systems. Based on the cost, measurement accuracy and comfort trade-offs of each motion tracking system, it was established that the best suited motion tracking system for the use case of this thesis would be the use of MEMS inertial sensors. Still related to motion tracking systems, various reference systems for representing orientation were also explored. Based on the strength provided from the quaternion representation (for not losing a degree a freedom when two axis overlap on each other), it was decided to adopt this representation in the developed work for calculating the respective knee flexion/extension angle.

Subsequently, a comparative analysis between different frontend application types was performed, exploring their strengths and weaknesses. From this comparative analysis it was concluded that for patients a mobile application would be best suited and for a physiotherapist a PWA, since it provides them with the versatility of viewing the application both on desktops and mobile devices. In this section, the backend development architecture is also debated, including what real-time protocols and notification systems should be used in the system, while also detailing what could be the potential wearable motion sensor infrastructure, as well as what network protocols to use for transmitting the sensor's knee angle readings.

Regarding the wearable sensor infrastructure, it was concluded that two ICM-20948 IMUs would be used for calculating the desired knee joint angle, along with two feedback actuators for providing feedback to the patients. For transmission of the angle readings, it was also established that between BLE and Wi-Fi network protocols, BLE had the most strengths for the proposed prototype in chapter 3, due to being available for use without requiring an internet connection and also for its low power consumption properties.

As for the backend, it was established that a relational database would be best suited since there would be various table relationships in the system, requiring more complex queries to be performed, and in terms of backend architecture, it was decided to follow some software architecture and design patterns in order to construct a modular and extensible backend.

Lastly some commercial solutions were also analyzed, in order to gain some insights about existing rehabilitation systems, and also get inspiration for the developed work. From the analyzed solutions, it was concluded that most applications had a mobile application component to support rehabilitation process and for also providing visual feedback, while the methodology of capturing movement metrics was different in each. Some solutions captured metrics by having support of sensors, while others were merely informative and displayed information about exercises.

3

KNEEMOR - Physiotherapy System

This chapter provides a description of the proposed system and explains the underlying motivation and the necessary system and user requirements in order to build an autonomous remote physiotherapy system.

The chapter starts by discussing the motivation and system requirements. In this section, the baseline of requirements that a remote physiotherapy system should have is explored, along with the necessary functionalities and features that must be implemented to meet the needs of each user type: physiotherapist and patient.

Afterwards, the system architecture and each of its components will be presented. In this section, the way each component interacts with one another using their respective communication protocols is discussed, as well as the way data will be stored and served throughout the system and how authentication and authorization is implemented on the system in order to segregate the respective actions for each user type. Finally, the proposed wearable sensor system will be introduced, detailing its core components and actuators for providing interaction between the patient and with the rest of the system.

3.1 Overview

As investigated in previous chapters, due to the pandemic and also since computers and mobile phones are ever more present in everyone's daily lives, more and more businesses and services and in particular health services, are moving onto online and remote solutions in order to keep providing their value at distance and also reduce operational costs.

Physiotherapy is a medical practice that typically requires the physical intervention of a physiotherapist on the patient, let it be to assess his current condition, guide him in rehabilitation exercises, provide feedback on the performance of exercises and also being responsible for prescribing rehabilitation treatments and protocols that are followed up and adjusted if needed in posterior consultations.

With this in mind, comes the proposal of ***KNEEMOR*** an autonomous physiotherapy and monitoring system which aims to provide typical health care system capabilities (appointment management, patient management, medical record keeping) but with an

emphasis on the medical practice of physiotherapy.

The system is targeted for the use of patients and physiotherapists alike, but its underlying user interfaces differ for each type of user.

As such, the system's main purpose on the patient's perspective, should be to promote knee rehabilitation through the means of an interactive mobile application that measures and monitors the knee joint angle in conjunction with a wearable sensor prototype. The wearable sensor prototype will be responsible for measuring the range of motion of the knee joint by using two IMUs: one placed on the posterior side of the thigh and the other placed on the posterior side of the shank. The wearable sensor prototype is prepared to address three types of physiotherapy exercises: heel slide, standing knee flexion and sitting knee extension. When undergoing an exercise, the wearable sensor prototype provides with visual and audible feedback through two actuators, a LED Bar to give patients an idea of how well they are doing the exercise and if they are meeting the specified upper and lower boundaries defined by the physiotherapist in range of motion and also a buzzer to provide an audible queue of when the exercise has finished.
The mobile application should prompt remainder notifications for executing the prescribed protocols and exercises, provide guidance to patients on how to perform them in a proper manner through visual and audible feedback, allow patients to access their current progress in range of motion in the knee joint through visual charts and reports, allow them to view and book medical appointments and lastly allowed them to request assistance or ask questions remotely via a chat system.

On the physiotherapists point of view, the system offers a single page application (SPA) with progressive web app (PWA) capabilities to provide both a mobile and desktop experience to physiotherapists. The physiotherapist's application should provide as previously explored in previous chapters, common health care system functionalities like appointment and patient management, keep a track record of each patient's progress in range of motion from the beginning until the end of the therapy, allow creation and assignment of protocols and exercises to patients as well as define their respective schedules, define their availabilites and unavailabilites for appointments, view and export patient's range of motion data through medical reports, and also allow them to communicate with patients through chat to answer any concerns that they might have.
When creating protocols and exercises the system allows physiotherapists to attach an image and also a video that can be used to give a preview of the exercise to patients, this allows patients to have an idea of the exercise they are performing and if they are doing it in the correct form.

Patients and physiotherapists in the proposed system, always belong to a Clinic which can be viewed as a medical facility that aggregates all the systems entities and data.

Proposed the system was tested in a local environment due to time constraints and the overall systems complexity, despite this some experimentation was done for cloud deployment but wasn't finished and fully implemented, leaving this particular point for future work.

3.2 System Functionalities and User Requirements

As explained in the overview, the system is comprised of two main stakeholders i.e. **patients and physiotherapists**. Based on these stakeholders, the proposed **features and requirements to be implemented** in the system will be explored in this section.

3.2.1 Patient Requirements

Patients, who are the main stakeholders in this system, are users that will undergo a rehabilitation process through the prescribed protocols and exercises made by physiotherapists.

Since the mobile application aims to provide an autonomous rehabilitation experience for the patients, **it should be intuitive and easy to work with**, meaning it should provide the necessary means to visually guide the patients through the pairing process of the wearable sensors, assist them when undergoing a particular exercise by displaying relevant exercise insights and the application should provide easy navigation and interaction in its various views. To adhere to these generic patient requirements, a list of more refined **features and requirements** were then established.

View assigned protocols and exercises. A patient should be able to view all his assigned protocols and their respective exercises. A protocol comprises one or more exercises by which a patient can view in detail (description, image and video) and also has some physiotherapist guidelines to follow during rehabilitation.

View exercise schedules. Upon selecting an exercise, the user should be able to view the respective schedules for that exercise. A schedule comprises a day of the week, a start hour and an end hour by which he should have performed the exercise. Also the patient should be able to view his weekly progress of the exercises he: performed, will perform or missed.

View and book appointments. The patient should be able view his appointments and corresponding appointment data, as well as be able to book future appointments by selecting a physiotherapist and picking the respective date and available time slot. Also the patient should be able to cancel a pending appointment and provide a reason for the cancellation.

Send and receive chat messages. A patient should be able to chat with his physiotherapists whom have approved their appointment bookings.

Receive reminder notifications for upcoming appointments and exercises. The mobile application should trigger appointment and exercise reminders at the end of the day to the patient so that he can still undergo a missing exercise or be reminded that he has an appointment in the next day.

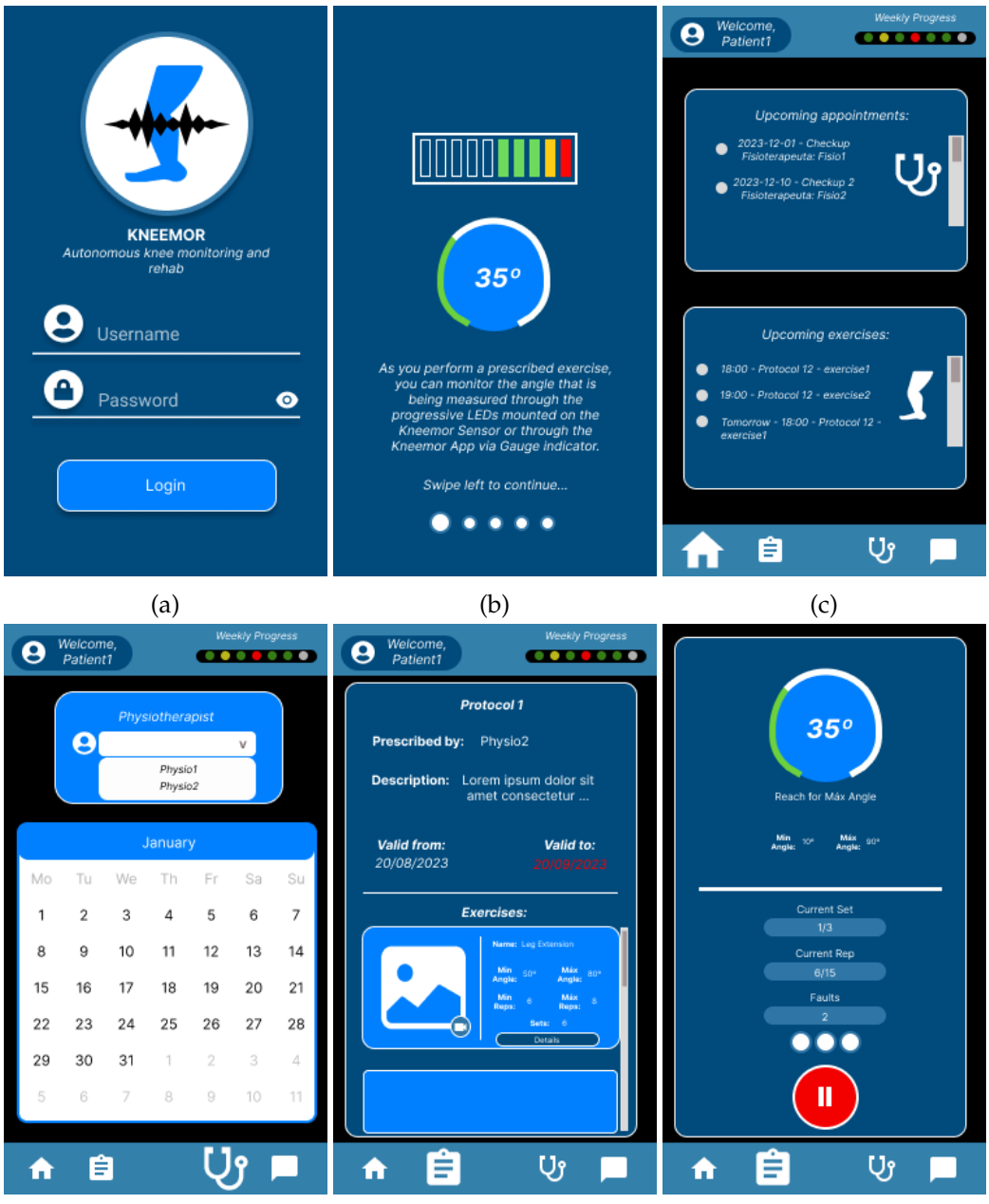
View and monitor his range of motion progress. The patient should be able to view his overall progress in range of motion of his knee as well as the faults he made in exercises through available reports.

Pair the wearable sensor prototype with the mobile application. The patient should be able to place the wearable sensor prototype correctly without any impairment and also be able to pair it with the mobile application via BLE so that the patient can start performing an exercise and the sensor data can be recorded. During the pairing process, the application should guide the patient visually through the various tutorial steps as well as the explaining the calibration step.

Start an exercise. When the sensor prototype is paired with the mobile application and calibration has ended, the patient should be able to start the exercise and through the application obtain the angle that is being read through sensors as well as access his exercise performance through an interactive user interface (UI) and also a visual and audible feedback provided from the wearable sensor prototype.

Export patient data and appointment data as a report. A patient should be able to export both its exercise data as well as appointment data in different file formats (.xlsx, .csv, .pdf).

During development some initial mockups of the patient views were created, to provide a path for implementing the various interfaces as depicted in Figure 3.1.



(a) (b) (c)

(d) (e) (f)

Figure 3.1: Mockups made with Figma comprising some screens for the patient's application. (a) Login screen, (b) Application Tutorial screen, (c) Home screen, (d) Appointment Booking screen, (e) Protocol screen and lastly (f) Exercise screen.

3.2.2 Physiotherapist Requirements

Physiotherapists on the other hand, are the other stakeholders for the system. On the physiotherapist application, as previously explored, the system should inherit some of the essential functionalities that a health care management system typically has. Additionally, since the physiotherapist application is visually adapted for both a desktop and a mobile phone, it should be responsive as much as possible with the main emphasis being on the desktop interface. Based on the previous research and on the operational necessities that a physiotherapist has, a list of **requirements and features** were established:

View and manage patient's details. A physiotherapist should be able to view all patients that belong to his clinic as well as edit their personal information.

Create exercise types. A physiotherapist should be able create a new exercise type which holds the name of the exercise, a brief description and associated media content (a image and/or a video). This exercise type can then be used to create exercises with their underlying parameters.

View, create and assign protocols and exercises. A physiotherapist should be able create a new protocol and assign it to patients, as well as define what exercises that protocol should have. Upon assigning an exercise to a protocol, the physiotherapist should be able to also set the underlying exercise schedule for patients.

View and manage appointment availabilities and unavailabilities. A physiotherapist should be able to manage his appointment availabilities as well as define his unavailabilities. In case the physiotherapist wants to clear a previous set unavailability, he can also delete that unavailability.

View, approve and reject appointments. A physiotherapist should be able to approve or reject pending appointments from patients and provide a reason for the rejection.

Attach appointment data to appointments. A physiotherapist should be able to attach multiple appointment reports to an appointment. A appointment data has a summary, diagnostic, treatment and recommendations for the patient.

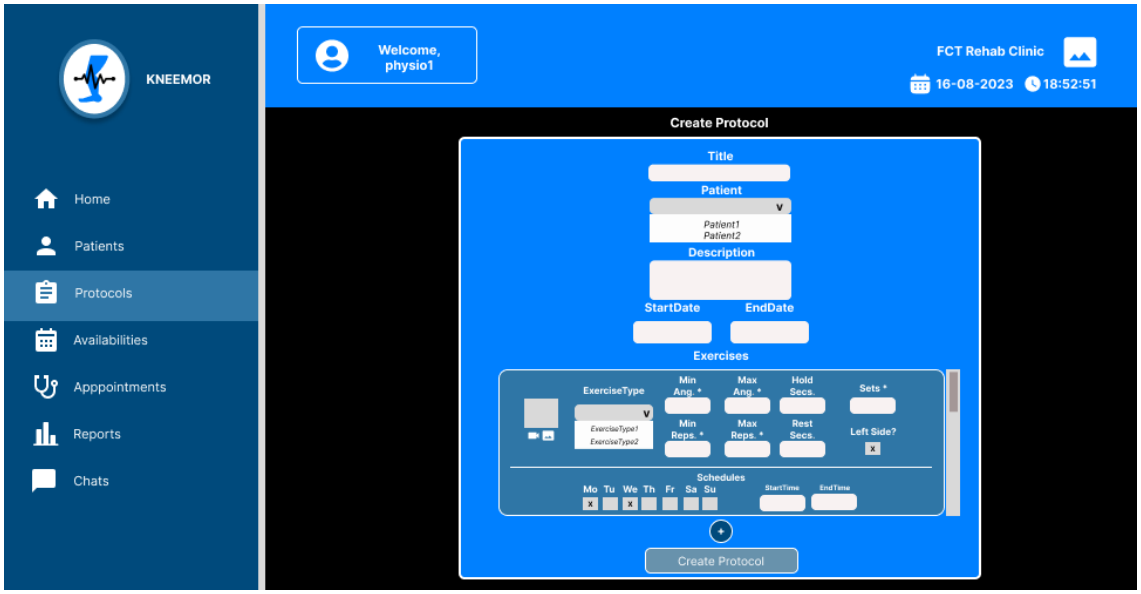
Send and receive chat messages. A physiotherapist should be able to chat with his patients, whom booked appointments with him.

Receive remainder notifications of upcoming appointments. A physiotherapist should be reminded of upcoming appointments he might have in the next day.

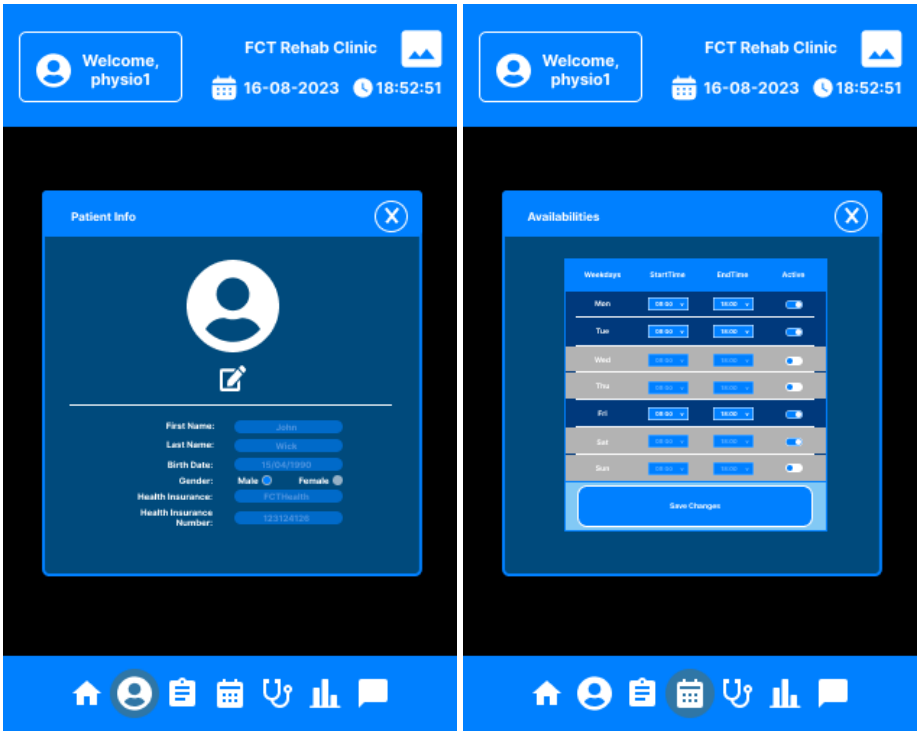
View patients progress through graphical views. A physiotherapist should be able to view the range of motion progress of a patient through visual graphs.

Export patient data and appointment data as a report file. A physiotherapist should be able to export both patient data as well as appointment data in different file formats (.xlsx, .csv, .pdf).

Some initial mockups were also designed for supporting the later implementation of the physiotherapist application as seen in Figure 3.2.



(a)



(b) (c)

Figure 3.2: Mockups made with Figma comprising some screens for the physiotherapist's application. (a) Protocol Creation screen in Desktop View, (b) Patient Information screen in Mobile View, and (c) Appointment Availability screen in Mobile View.

3.3 Proposed System Architecture

Based on the research performed on chapter two, and after exploring the underlying technologies, there's now a better understanding on how to create a remote rehabilitation system from the ground up that leverages the goals established for the elaboration of this thesis.

The system architecture should be comprised of three main layers: **a sensor layer, a frontend layer and a backend layer** as depicted in the Figure 3.3.

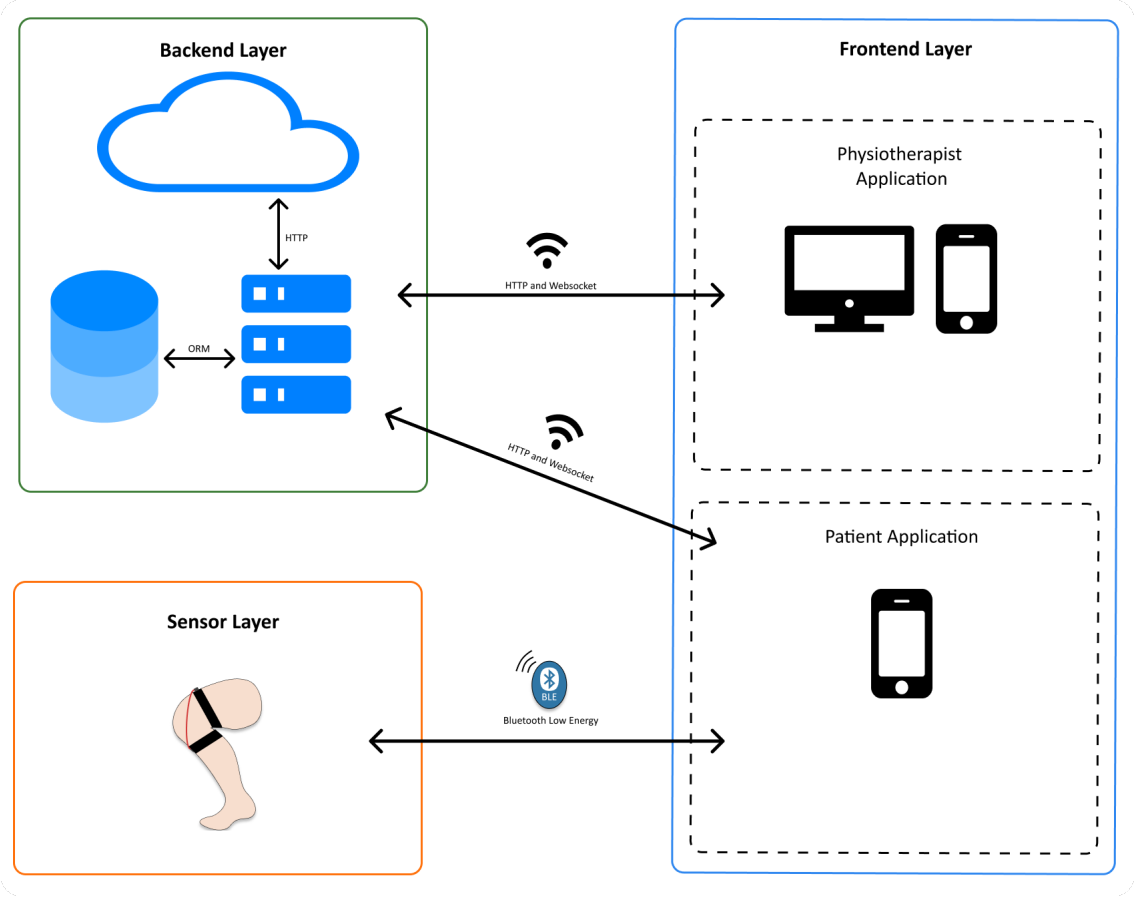


Figure 3.3: System Architecture

The sensor layer comprises a wearable sensor prototype which is responsible for measuring the knee range of motion, provide visual and audible feedback when the patient is performing the exercise and is also responsible for relaying the measurement data to the patient's mobile application.

The frontend layer is split into two distinct applications, an **mobile application** that is tailored to the patient requirements and a **progressive web app** which attends the physiotherapist's operational needs.

Both applications, despite being distinct, have a similar design look and feel, as well as similar codebases to accelerate the development process, but only the patient's application interacts with the sensor layer and the backend layer, leaving the physiotherapist

application only being able to interact with the backend to retrieve system information. For interacting with the backend layer it is assumed that both applications support WiFi in order to perform requests and retrieve data from the backend.

The backend layer ensures data storage of the system and provides the access to its data to both frontend interfaces through an REST API. This layer also interacts with a cloud service for storing blob files that will be explored later on. In addition the backend provides an endpoint to establish a Websocket connection so that chat messages flow between both frontends.

To achieve this architecture, it is important to experiment and explore the available technologies and libraries that can be used to implement all these layers as well as provide a list of minimal requirements per layer.

3.3.1 Sensor Layer

The sensor layer, as stated previously, comprises the wearable sensor prototype which happens to be the main component and subject for this thesis. It ensures the measurement of the knee angle when a patient is performing a physiotherapy exercise and that these measurements are correctly relayed to the other layers.

Based on the previous research, it was concluded that in order to achieve a knee angle range of motion measurement with some accuracy, two IMUs would be necessary as well as the respective feedback actuators to provide guidance when performing an exercise and a ESP32 microcontroller to provide communication and processing between all these components.

As for transmitting the measurement data, it was studied that the best approach would be using the BLE network protocol, as it provided the best choice in terms of power consumption and offline support.

The IMUs placement that was chosen was one IMU along with the microcontroller and its actuators on the anterior side of the thigh right above the patella due to its wider area to support all the components, and the second IMU on the anterior side of the shank right bellow the patella. Based on experimentation, this placement had the best results in terms of measurements and also works for all the types of exercises that were proposed.

Having set the hardware requirements for the prototype, we must also guarantee that a few **functionalities and requirements** are met, so we can provide the best rehabilitation experience for patients. With that in mind the following requirements are established:

Cost effective. The wearable prototype should have affordable components so it has a bigger reach for both physiotherapists and patients alike.

Comfortable and easy to wear. The wearable sensor prototype should be easy to apply on the affected knee without direct intervention of a physiotherapist. It should be non-invasive and as less constricting as possible for knee movement so that the physiotherapy exercises can be conducted without major movement restrictions.

Provide audible and visual feedback. The wearable sensor prototype should have feedback actuators that can guide the patient on the movements he performs and also notify him when an exercise has concluded.

Perform knee angle measurements and send data to the patient's application. The wearable sensor prototype should be able to measure the knee range of motion in terms of flexion and extension, provide the angle measurement as well as raw quaternion data to the patient's application so it can be stored in the database and accessed later for reviewing results and progress by the physiotherapist.

The conceptualized Sensor Layer, hardware wise is composed of: the **ESP32 WROOM DEVKIT-C, a electromagnetic buzzer, a Grove LED bar and two ICM-20948 IMUs** as seen in Figure 3.4. The choice for an IMU with 9 degrees of freedom is due to the fact that it can leverage a better sensor fusion (mixing the data from various sensors) in comparison to a IMU of 6 degrees of freedom and for offering the on-chip sensor fusion which relieves computational power from the microcontroller [6]. The ESP32 microcontroller was chosen due to offering integrated WiFi and Bluetooth modules, being relatively cheap and providing programmable pins which were crucial for the prototype development.

For feedback actuators, a buzzer and a Grove LED bar are more than enough to provide visual and audible feedback to the patient. These actuators indicate if the patient is trespassing the designated lower and upper thresholds for the range of motion and also to indicate when an exercise ends.

Programming wise on the microcontroller, there should be a BLE Server to provide interaction between the mobile application and the sensor prototype and the microcontroller should also provide good libraries for configuring and programming the underlying hardware components. For this task, Arduino programming language was chosen.

3.3.2 Frontend Layer

This layer provides the overall interaction between the stakeholders and the rehabilitation system. The mobile application should comprise the main features that were previously discussed for patients and the data access should be through the REST API that is available on the Backend Layer.

The patient's mobile application also serves as a bridge between the prototype sensor for initiating a exercise and stopping a exercise. The mobile application also displays visual feedback for the patient with the readings sent by the sensor and if there is enough time during the elaboration of the thesis, there might also be an interactive UI that draws a 3D leg and maps the sensor readings to that model moving it interactively.

As for the physiotherapist application, the interactive UI is not necessary for the physiotherapists, but instead there's a need of analytical dashboards that comprise all the patient's health data, scheduled appointments, protocol and exercise configuration and export data.

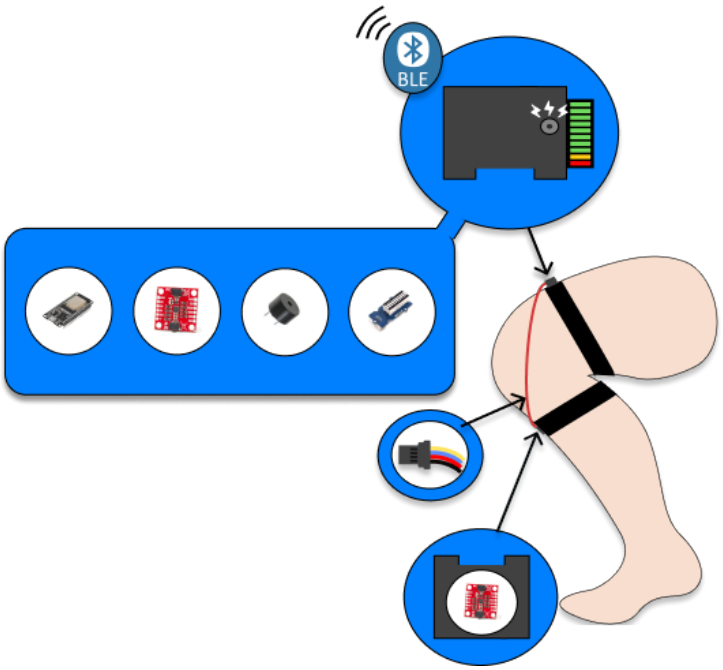


Figure 3.4: Wearable Sensor Prototype, composed of a ESP32 microcontroller, two ICM-20948 IMUs connected with a Qwiic cable, a buzzer and a groove LED bar.

For the actual implementation and programming of both applications, Typescript will be used which is a subset of Javascript, it provides type-safety which prevents potential bugs from happening and promotes to avoid type inference since it can originate in incorrect typing that produce unexpected results along the development. In conjunction with Typescript, the React library will be used since it is capable of building interactive UIs and provides a mobile counter-part framework which is React Native. By using React we can have the same codebase for both the patient's application and the physiotherapist's application throughout the development which in turn helps in time-saving and gives an overall better developer experience. Auxiliary libraries for dealing with several concerns will be used during development and will be discussed in the implementation chapter with more detail.

3.3.3 Backend Layer

The Backend Layer as stated previously, is the gatekeeper of all the system's data and provides a web API interface to both frontend applications so both of them can query and store relevant system data.

This layer comprises a database and a backend server which exposes a REST API that follows REST semantics and standards on its underlying endpoints, and each endpoint is dedicated to a specific feature or concern. On the backend server there is also a specific endpoint exposed for establishing a Websocket connection with a frontend client. This Websocket connection provides a communication medium for the chat system where physiotherapists and patients can chat as long as an appointment has been accepted and

involves both parties. For storing exercise media and profile pictures for the patient and physiotherapist a blob storage solution on the cloud should be used.

Having these base requirements in mind, a set of **features and functionalities** are then expected from the backend layer in order to support the system as whole.

Modular and easy to maintain. The backend server should be segregated into different projects following the Clean Architecture pattern, promoting low coupling and high cohesion between each layer's responsibilities. The various features of the system should be well isolated and offer abstractions so that implementations can be easily swapped without affecting the overall backend structure.

Open-source and support for series generation. The backend server as well as the database should use open-source technologies for better system longevity and transparency, and the used database should support series generation, so that appointment time slots can be generated dynamically without the need of storing said data in the database.

Well structured REST API abiding REST standards and semantics. The backend REST API should follow REST standards in terms of endpoint definition and also use the correct HTTP methods and status codes depending on the type of requests that are being made.

Provide authentication and authorization of the underlying resources. The backend server should authenticate and authorize requests depending on the type of user that is making the requests to the backend server. The authentication process should be applied both on the REST API as well as on the chat system it provides.

Provide data secrecy. Due to the particular nature and area of this system, personal data of patients and physiotherapists should be anonymized and only viewed through the underlying applications.

Provide reports of patient data and appointment data. The backend should expose special endpoints that allow for generation of reports about the system's data, using well defined templates that have a feel and appearance of typical medical reports from medical facilities.

Provide external interface for the chat system. The backend server should expose a special endpoint in order to establish a websocket for the underlying chat system and the requesting user. Through this channel chat messages are flowed in and out of the backend layer and to the respective frontend applications.

Send reminders for appointments and exercises. The backend server should send at the end of the day reminders for upcoming appointments and exercises to the users that are currently connected to the system via websocket. This feature might not be implemented due to time constraints and to the overall complexity of the system as it is.

As for technological stack, the backend language being used should be mature, strongly typed and well adopted in the industry and also offer a underlying backend framework comprising the necessary services and functionalities that are required for the backend layer construction.

As for database, it will be comprised by the following entities: **Clinic, User, UserRole, Role, Patient, Physiotherapist, Protocol, Exercise, ExerciseType, ExerciseSchedule, ExerciseCompletion, ExerciseReading, Appointment, AppointmentData, Availability, Unavailability, Chat and ChatMessage**.

Authentication details are related to the **User, UserRole and Role** tables. The User table holds log in information and also common properties for both Patients and Physiotherapists.

The **Physiotherapist and Patient** tables have specific information related to each type of user and both entities can interact with Appointments, Chats and Protocols.

A **Protocol** comprises one or more **Exercises**, and an **Exercise** has a specific **ExerciseType and one or more ExerciseSchedules**.

ExerciseCompletions and ExerciseReadings are related to a particular **Exercise and Protocol**. These two entities define if a patient has concluded an exercise on a give date and hour and also stores the sensor data that was emitted by the wearable sensor prototype.

Appointment and AppointmentData are the entities that are responsible for medical appointments and for accessing their underlying appointment's data.

Availability and Unavailability tables are the entities that define the physiotherapist's appointment schedule and which provide the available time slots and days that patients can perform appointment booking.

Lastly, the **Chat and ChatMessage** tables are used for storing created chats and also for storing chat messages related to a chat between a physiotherapist and a patient.

For the database choice, based on the previous table relationships and requirements, it should be a relational database and in this case the database engine should be **PostgresSQL** for being open-source and having support of series generation. The deployment of the application will be explored but won't be the main focus of this thesis, leaving space for future work.

3.4 Conclusions

The *KNEEMOR* system should be composed of three respective layers: a sensor layer, frontend layer and backend layer, each having their own set of specific responsibilities. The frontend layer should be split into two frontend applications, a mobile app and a progressive web app, where the first concerns the patient's needs and requirements, and the latter concerns the physiotherapist's daily medical operations. The system as a whole has the main objective of promoting the rehabilitation process for patients by combining the mobile application with the prototype, whilst providing monitoring and assessment of a patient's condition to both physiotherapists and patients alike, through the means of visual interfaces and also downloadable reports.

4

KNEEMOR - System Implementation

This chapter documents the final system implementation based on the set of requirements and functionalities that were presented in the proposal chapter. Here it is addressed in detail the various system layers, the architecture, features implementation and the various libraries and tools used for developing each functionality and bridging the different layers.

In the end of this chapter a summary is presented summarizing the implementation chapter and presenting the final system architecture already with the various technologies in place.

4.1 Sensor Layer

The Sensor Layer as seen in the previous chapter contains the core component of this thesis which is the wearable sensor prototype. This prototype plays a pivotal role in the *KNEEMOR* system, as it is responsible for collecting, processing and delivering knee angle data when undergoing a physiotherapy exercises. This collected data is then sent to the patient's application, where it can be visualized in real-time and uploaded afterwards to the backend for future consultation and analysis from physiotherapists. In this chapter, the implementation details of the prototype are discussed and also how the design process was tackled.

4.1.1 Wearable Sensor Prototype Design

Based on the thesis proposal, the prototype had the requirements of being comfortable, providing visual and audible feedback, providing the necessary knee angle readings and providing transmission capabilities to the patient's application. The initial prototype was assembled following the schematic depicted in Figure 4.1, consisting of an **ESP32 WROOM 32D DevKitC development board**, two **Sparkfun ICM-20948 IMUs**, a **Seeed Grove LED Bar** and an **electromagnetic buzzer**. The two IMUs were daisy chained through a **Qwiic Cable** and share the same I2C bus for transmitting their sensor readings by different IMU chip addresses. The IMU not connected to the breadboard was placed in the anterior side of shin and has the address jumper soldered (the address value is 0), and

the other IMU is placed on the anterior side thigh along with the rest of the components (and had the default address, which is 1). It is important to note that these IMU positions are crucial for the prototype, since the underlying calculations for extrapolating the knee joint angle require these specific placements to obtain correct angle readings, but the sensor placements works for either leg. The thigh's IMU powers the shin IMU by being connected to the ESP32 through the I2C pins (SDA [IO21] and SCL [IO22]), GND pin and 3.3V pin. The Grove LED Bar was connected to the development board through a grove cable which was also connected to GND and 3.3V pins and also the IO0 and IO4 ADC pins. Lastly the buzzer was connected to the IO2 ADC pin and also to the GND pin. Due to the feedback actuators sharing the GND and 3.3V pins with the IMUs, it was necessary to solder some cables together in order to facilitate the cable management in the final prototype assembly.

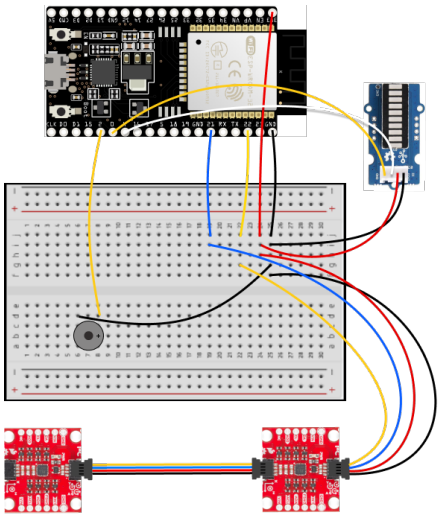
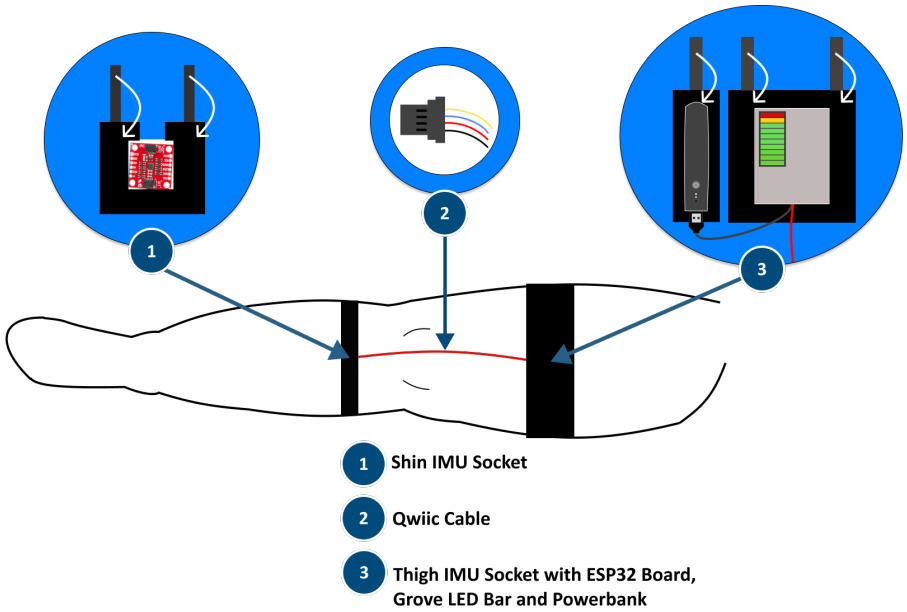
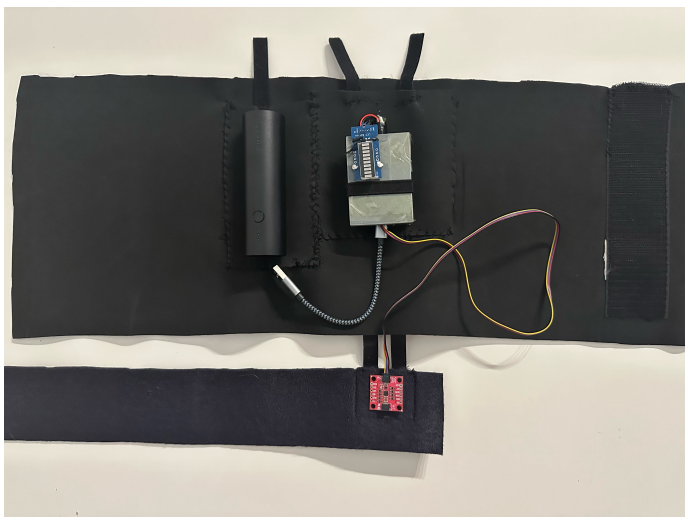


Figure 4.1: Wearable Sensor Prototype - Schematic

Also, during the prototype's development and testing, it was detected that the buzzer caused interference with the IMU readings. For this reason, and due to time constraints, it was removed from the final design but the audible feedback was maintained by the mobile application through sound effects. In the final design, the wearable sensor components had to be properly accommodated and secured to prevent any friction and discomfort while performing the knee physiotherapy exercises. To accomplish this, a malleable and elastic waistband was used and adapted to make two detachable leg sleeves with sockets that could be closed with velcro straps. One socket was used for holding the shin's IMU and other was used to holding a 3D printed box containing the thigh IMU, development board, the Grove LED Bar and a power bank for providing power, as seen in Figure 4.2. The decision to use the thigh for holding most of the components was due to its anatomical coverage area being superior to the shin area, making it ideal for holding most of the components without causing too much strain on the leg when doing physiotherapy exercises.



(a)



(b)

(c)

Figure 4.2: Wearable Sensor Prototype - Final Design. (a) Final schematic, (b) Detailed view of prototype's components placement in leg sleeves, (c) Prototype mounted on leg

Finally, due to the ESP32 development board offering various onboard functionalities, for transmitting the processed sensor data to the patient's application, the BLE protocol was used due its low power consumption and also because it does not require an internet connection to send the sensor data to the patient's application.

4.1.2 Sensor Initialization Process and Payload Transmission

After going through the possible communication protocols for providing the data transmission between the prototype and the patient's application, it was established that BLE had the most leverage and advantages for the current use case, due to its low power consumption, offline support and also for the fact that ESP32 provided this onboard module facilitating the prototyping.

The wearable sensor prototype for transmitting the IMU readings to the patient's application required the setup of a BLE Server, offering a service with four underlying characteristics: **the setup characteristic, the status characteristic, the data characteristic and the LED characteristic**.

The **setup characteristic** is responsible for providing the sensor initialization process to the patient's application. This characteristic sets up the maximum angle threshold for the progressive LEDs, allows the patient application to select the output mode of the sensor and also allows the starting/stopping of the calibration phase. These various settings are defined through a write operation on the characteristic, and depending on the value that is read from **status characteristic**, the next step to follow in the initialization process is specified.

The **data characteristic** is responsible for delivering the sensor data readings in the specified output mode. It provides the values through a notify operation.

The **LED characteristic** specifies a write operation for setting the LED state for when an exercise is complete.

Before undergoing a physiotherapy exercise, an initialization process must take place as depicted in Figure 4.3.

The communication process between the wearable sensor prototype and the patient application starts when the sensor prototype is turned on and the broadcasting phase begins.

In this phase, the sensor prototype becomes discoverable and awaits a connection establishment from the patient application. When beginning an exercise on the patient application, a prompt is issued to the patient to initialize the scanning for the *KNEEMOR* sensor. If the sensor is detected then the output selection phase begins.

In the output selection phase, the patient application sends the exercise maximum angle first, for setting up the progressive LED Bar maximum threshold, and afterwards the user is prompted to choose the output mode of the sensors.

In the developed work, the sensor prototype offers three possible output modes: **Angle mode (A), Quaternion mode (Q) or Both mode (B)**.

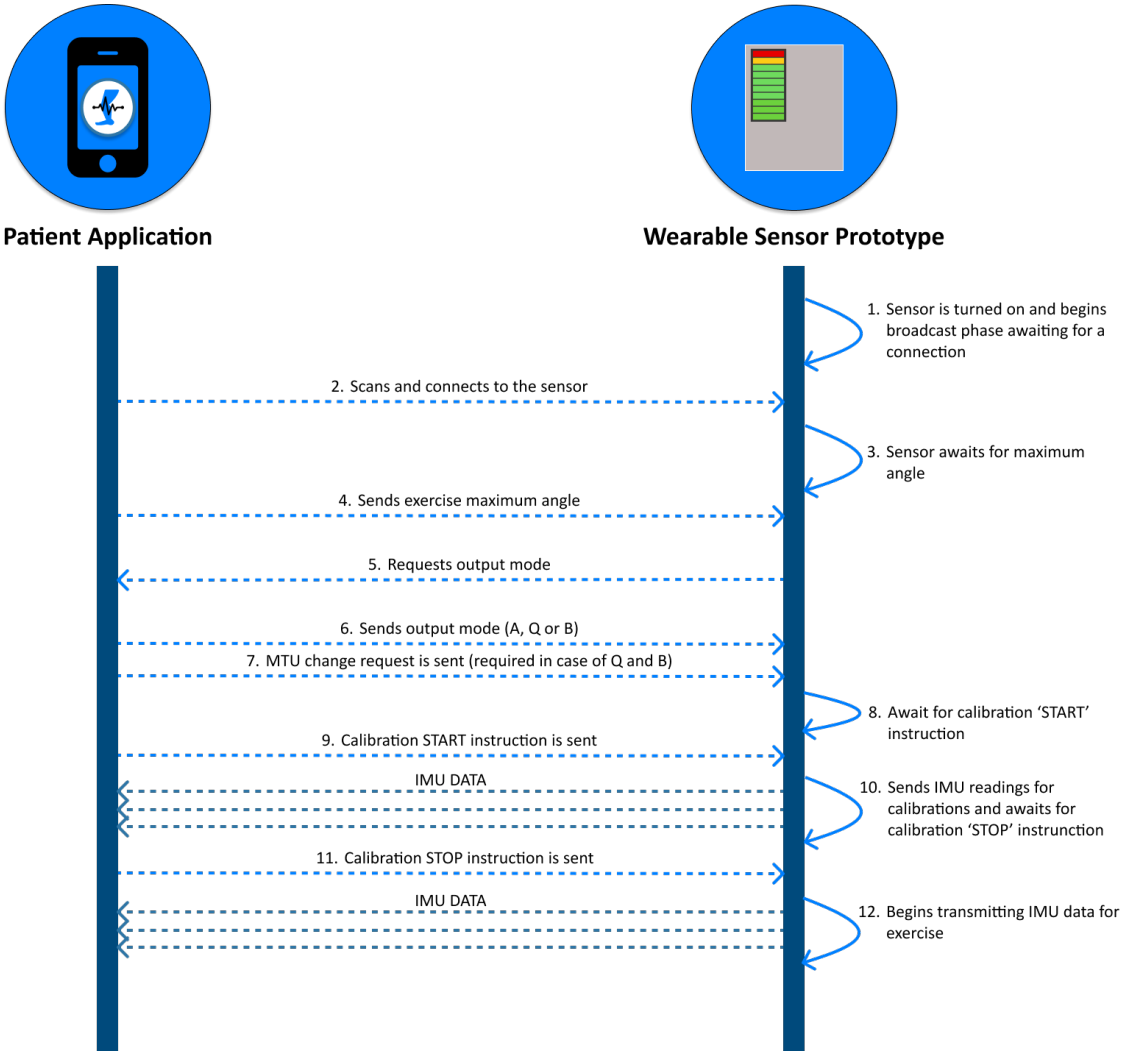


Figure 4.3: Sensor Initialization Process

Depending on the output mode that is selected, the provided data payload size differs. For the **Angle mode (A)**, only the knee angle is provided to the patient application. The payload supports the range of values from [-999.99 ; 999.99] and its size is defined by 1 byte for the sign, 3 bytes for the integer part, 1 byte for the decimal point, 2 bytes for the fractional part and 1 byte for null terminator, resulting in a total payload of 8 bytes.

The **Quaternion mode (Q)**, provides each sensor's orientation in a quaternion format for the patient's application. This mode was implemented with the idea of supporting a 3D representation of the leg movement whilst undergoing an exercise.

Each quaternion component is bounded in a range from [-1.00 ; 1.00], and the decimal precision of each component being 2 decimal places, for each quaternion part it required 6 bytes (1 byte for the sign, 1 byte for the integer part, 1 byte for the decimal point and 2 bytes for the fractional part). Since one quaternion has 4 parts (x, y, z representing the unit vector and w for representing the scalar part), the total bytes necessary for the two quaternions is 5×8 bytes = 40 bytes. However, we must also have a separator between

each value, so we know what quaternion part is referred to, and in this case in the payload size we also require 7 bytes for semicolons. When all the values are appended to each semicolon, we must also use 1 more byte for the null terminator, resulting in a 48 byte long payload.

The **Both mode** combines both payload sizes, by appending 1 more semicolon at the end and also the angle, resulting in a 56 byte long payload.

An important note is that when using the last two output modes, the Maximum Transmission Unit (MTU) size has to be changed accordingly, since the default MTU size typically is 23 bytes long. On the BLE server from the sensor, we set the maximum accepted MTU size to 56 bytes (which corresponds to the largest payload size) by using **`BLEDevice::setIMU`** instruction. In the patient application, when one of these output modes are selected, we must additionally issue a MTU change request, indicating the respective payload size, otherwise the transmitted data will not be fully complete.

After selecting the output mode, the sensor prototype awaits for a **`START`** instruction from the patient's application in order to start the calibration process on the patient's application. The ICM-20948 IMUs already perform some internal calibration for us but since there are different leg anatomies for different types of people, an additional calibration is made on the patient's application in order to calculate the offset value that must be taken from the IMU readings.

Once the calibration is performed, the patient application must send a **`STOP`** instruction to the sensor prototype, so the exercise can then begin.

When the exercise is completed, a **`COMPLETED`** instruction is sent to the LED Characteristic, causing the progressive LEDs to blink for a period of time and indicating the exercise has been completed. After this animation is completed, the sensor state is reset and returns to its broadcasting phase until another exercise is selected.

4.1.3 Knee Joint Angle Calculation

In the developed prototype, to calculate the knee joint angle a quaternion based approach was used due to its strengths in representing 3D rotations and also for not suffering from the gimbal lock problem as seen in the state of the art section [59].

The two quaternion readings provided from the two ICM-20980 IMU's digital motion processors sensor fusion, provide for each quaternion a 9DOF orientation of them.

By having each IMU orientation, we can extrapolate the relative orientation between the two IMU orientations by using quaternion properties and multiplication.

The relative orientation between two quaternions is given by the following formula:

$$q_{RelativeOrientationShin} = q_{shin} * q^*_{thigh}$$

Where $q_{RelativeOrientationShin}$ is equal to the multiplication of q_{shin} which is the quaternion provided from the IMU placed on the shin by the conjugate of the quaternion provided by the IMU placed on the thigh q^*_{thigh}. The resulting quaternion provides us

with the relative orientation of the shin in respect to the thigh. If we switched the order of the quaternions in the formula into $q_{thigh} * q^*_{shin}$ we would instead get the relative orientation of the thigh in respect to the shin. Since we are interested in calculating the knee joint angle, we must know how much the shin has rotated relatively to the thigh, hence we use the first order in the multiplication.

By having the relative orientation of the shin in respect to the thigh, we can now convert the resulting quaternion into the respective Euler angles (Roll, Yaw, Pitch) by using the formulas studied in chapter two [59].

$$\begin{aligned} \text{Roll}(\gamma) &= atan2(2 * (q_w * q_x + q_y * q_z), q_w^2 - q_x^2 - q_y^2 + q_z^2) \\ \text{Pitch}(\beta) &= asin(2 * (q_w * q_y - q_x * q_z)) \\ \text{Yaw}(\alpha) &= atan2(2 * (q_w * q_z + q_x * q_y), q_w^2 + q_x^2 - q_y^2 - q_z^2) \end{aligned}$$

Since in the developed work we are only interested in the extension/flexion angle of the knee joint, which is the rotation that happens around the Y axis, the Euler angle to use in this case, is the pitch angle as seen in Figure 4.4.

However, during development it was discovered that the roll angle was giving the correct readings and the pitch angle was providing erroneous values. The potential reason for this behavior could be that the local axis references of each IMU could be different from the standard conventions like presented in Figure 4.4. If the X axis is perpendicular to the anatomical axis of extension/flexion of the leg and not parallel, the roll formula could in fact provide us the pitch angle instead. Nonetheless, due to this potential axis misalignment or chip defect, the roll calculation was instead used to provide the desired angle.

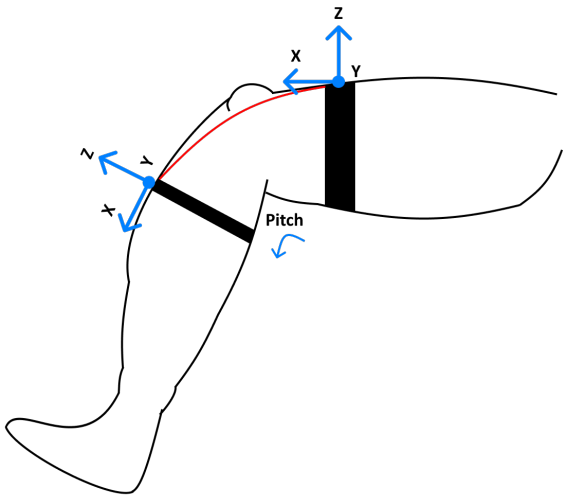


Figure 4.4: Median plane view of leg with the standard IMU axis references

4.2 Backend Layer

The backend layer is the backbone of any web application, responsible for handling critical responsibilities like data storage, data access, authorization and authentication, security, applying business logic rules and providing communication interfaces for various system components, both internal and external.

To establish this layer, as previously mentioned in the proposal chapter, we will need a backend server, a database and a blob storage service as main components.

For the backend server, ASP.NET Core 7 will be used, an open-source web framework for .NET flavoured languages (C#, F#, etc). PostgreSQL will be used as the database choice and lastly Azure Blob Storage Service for storing blob data. The criteria for adopting these technologies was based on personal preference and experience, as well as the various advantages mentioned in the second chapter, which make developing certain system features easier.

This layer should also be well-segregated, with different boundaries for different responsibilities. Having this separation of concerns in the backend layer promotes easier maintainability, scalability and extensibility, allowing for easier adoption of new features and reducing the system disruption when changes are made.

In this chapter, we begin by exploring the backend structure and organization based on separation of concerns and adherence to the Clean Architecture software pattern. Afterwards the database design process is explained, followed by the discussion of the REST API design and how each resource is accessed based on the authentication and authorization implementation. And lastly, it is briefly addressed how the Notification Hub provides the real-time chat system for physiotherapists and patients.

4.2.1 Backend structure and development libraries

Since in the proposal chapter it was established that the backend layer should be well-structured, modular and easy to maintain, for implementing the backend server a well-adopted software architecture pattern was chosen, namely Clean Architecture along with some design patterns, namely **Repository, Unit Of Work and Mediator patterns**.

To adhere to the Clean Architecture pattern, the .NET Core 7 backend solution encompasses 4 projects (or layers): **Core, Application, Infrastructure and Web.**

The **Core** layer as previously explored in the second chapter, encompasses data access objects, repository interfaces and external service interfaces. This layer defines database entities used in PostgreSQL and through the provided interfaces defines the available functions that outer layers should comply with. Through the entities or data access objects defined here, the table relationships are established, the column types as well as which columns should be anonymized. The repository and external services interfaces serve as method contracts for the implementation classes that reside in the Infrastructure project.

The libraries that were used to orchestrate this layer were the following: **EntityFrameworkCore.EncryptColumn** [1] which provides the capability of anonymizing columns in the PostgreSQL database, **Microsoft.AspNetCore.Http.Features** which allows leveraging the IFormFile interface that contains file metadata, useful for our IBlobStorage interface and lastly **Microsoft.Extensions.Identity.Stores** which allows for referencing the default Identity tables that are used for authentication and authorization purposes.

The Core layer as mentioned in the Clean Architecture, is the innermost layer and does not reference any outer layer, meaning it does not know about implementation details from outer layers and only supplies with the interfaces that are required for the outer layers.

The **Application** layer is where the business rules, features and external contracts reside, and where the main flow control of the backend application is handled. When incoming requests reach the backend's Web layer, these requests get wrapped inside messages that are relayed by MediatR, and are then handled by the Application layer.

MediatR[2] is the .NET implementation of the mediator pattern, serving as a central hub for sending messages between objects without having the need of these objects having a direct reference to one another. This promotes loose coupling between objects making the backend more maintainable.

The MediatR messages can be viewed as either **Commands or Queries**, where **Commands** represent database mutations and **Queries** represent database reads. The mediator pattern is often used in conjunction with the command query responsability segregation (CQRS) pattern which involves using two databases, one for executing mutations and another one for executing reads. In the developed work only one database was used, and instead the command query separation (CQS)[3] pattern was applied to separate the mutations and reads through the use of messages and message handler classes. For interacting with the Infrastructure layer implementations, message handler classes use the repository interfaces, which are injected with the respective implementations through .NET's dependency injection (DI), and through these injected services we can perform operations on the database without knowing the implementation details. Using this approach on our backend application we can easily swap database engines and implementations without breaking the remainder code which only relies on interfaces.

Also in the Application layer, is where various utility classes and exception classes are stored. These utility classes provide functionality for creating the data reports in different formats and also provide the JSON Web Token (JWT) builder class that is responsible for generating access tokens and refresh tokens for authentication concerns.

The Application layer only references the Core layer, since it requires access to the entity classes in order to map them into the respective DTO classes.

[1] EntityFrameworkCore.EncryptColumn - Third-party library for encrypting columns with EntityFrameworkCore - https://github.com/emrekizildas/EntityFrameworkCore.EncryptColumn

[2] MediatR - Simple mediator implementation in .NET - https://github.com/jbogard/MediatR

[3] Command-query seperation - https://en.wikipedia.org/wiki/Command-query_separation

The libraries that were used to orchestrate this layer were the following: **ClosedXML**[4] provides spreadsheet manipulation and creation, **Microsoft.Authentication.JwtBearer** which provides utility classes for creating JSON Web Tokens, **MediatR**, **AutoMapper**[5] for providing quick mapping between DTOs and DAOs, **QuestPDF**[6] for report generation in PDF format and **ScottPlot**[7] a chart library that can be combined with QuestPDF for generating reports that also comprise charts images.

In the **Infrastructure layer** is where the repository and external services implementations are located, along with the definition of the EntityFrameworkCore database context and its various tables. Through the database context and its table definitions based on the data access objects, we can use EntityFrameworkCore to generate the required SQL for creating the entire database.

Through the repository implementations, which also use the data access objects, queries are created programmatically to fetch data from the database using the EntityFrameworkCore object relational mapper instead of writing raw SQL instructions.

Additionally, in this layer is where we define the Azure Blob Storage service for interacting with our blob containers to upload patient images, physiotherapist images, clinic images and exercise media (image and video). Lastly for testing purposes, some seed configuration classes were defined. These classes basically seed each table from our database with some mock data. The generation of this mock data is done randomly using a .NET library for faking data named **Bogus**[8].

The Infrastructure layer directly references only the Application layer, and gains access to the Core layer transitively through this reference.

The libraries used to orchestrate this layer were the following: **Azure.Storage.Blobs** for manipulating blobs on Azure Blob Storage, **Bogus** for generating fake data, **EntityFrameworkCore.EncryptColumn** for decrypting and encrypting database columns, **Microsoft.EntityFrameworkCore** the object relational mapper for .NET built by Microsoft, **Microsoft.AspNetCore.Identity.EntityFrameworkCore** for providing access to the default Identity tables that are used for authentication and authorization purposes and lastly **Npgsql.EntityFrameworkCore.PostgreSQL** which provides the PostgreSQL database driver for EntityFrameworkCore.

The final layer, which serves as the point of entry to the backend server, is the **Web Layer**. This layer is responsible for registering all backend services, including authentication, authorization, additional middlewares, API endpoints, hubs and others. In the Web layer, service interfaces are mapped into their corresponding implementation classes by being registered through .NET's dependency injection container. The registration process is provided through the **IServiceCollection** interface, which is available on the entry point of

[4]ClosedXML - Open-source .NET library for reading, manipulating and writing Excel 2007+ (.xlsx, .xlsm) files - https://github.com/ClosedXML/ClosedXML

[5]AutoMapper - .NET library for mapping objects - https://github.com/AutoMapper/AutoMapper

[6]QuestPDF - Open-source .NET library for PDF generation - https://github.com/QuestPDF/QuestPDF

[7]ScottPlot - Open-source plotting library for .NET -https://github.com/scottplot/scottplot/

[8]Bogus - a fake data generator for .NET - https://github.com/bchavez/Bogus

the application. The services used throughout the backend application rely on interfaces to decouple their implementations, and thanks to dependency injection, whenever a class requires an interface that has been registered via IServiceCollection DI container, an instance of the corresponding implementation is automatically provided, allowing the class to use it without needing to know the implementation details.

Also regarding dependency injection, each layer (except for the Web and Core) contains a special class at the root level of the each project, which is named DependencyInjection. This class uses an extension method for IServiceCollection interface, enabling us to register each layer's services within their project boundary. In the Web layer, these extension methods are invoked and the registration of the underlying services is performed transitively for each layer.

In this layer is where the REST API endpoints, SignalR notification hub, and the background service responsible for pushing daily reminders are hosted. It's through this layer where both frontend applications communicate and where they perform requests to query and mutate data on the *KNEEMOR* system.

The libraries that were used to orchestrate the Web layer were the following: **Microsoft.Authentication.JwtBearer** for providing the JWT schemes and utility functions, **Microsoft.AspNetCore.Identity** an identity provider for authentication and authorization configurations, **Microsoft.AspNetCore.OpenApi and Swashbuckle.AspNetCore** for providing methods to generate OpenAPI documentation and providing also its visualization, **Microsoft.EntityFrameworkCore.Design and Microsoft.EntityFrameworkCore.Tools** for handling database scaffolding and migrations, **Serilog.AspNetCore** a logging provider with several log sinks.

4.2.2 Database design and implementation

The database design for the *KNEEMOR* system was inspired on the database modeling of a medical facility, more specifically a clinic. From the proposal chapter, various tables were conceptualized and some of their relationships were disclosed, but now we go into detail in these relationships and also how the database and its various tables were created using Entity Framework Core.

When using Entity Framework Core, we can choose two routes for implementing the database: a **database first (or scaffolding) approach** (in the cases that we have already a database created and only want to map each database entity into the corresponding C# entity class which enables database interaction), or a **code first approach**[9] (where we define all our database tables and their relationships through C# code, which then gets mapped into the necessary SQL instructions that get executed in the corresponding database engine).

[9]Code-First Approach - https://learn.microsoft.com/en-gb/ef/ef6/modeling/code-first/workflows/new-database

In the developed work, a code first approach was adopted due to its strengths in prototyping, simplifying and accelerating the whole database design process and also due to the fact that there wasn't any existing database previously to work with.

Through code first approach, a set of C# entity classes were created in the Core Layer (representing each database entity and its underlying fields), and in the Persistence Layer the database context was created.

In the C# entity classes, annotations are used to define what properties each specific column should have. These annotations are capable of defining primary and foreign keys, column types and sizes, nullability, and also define relationships. Primary keys can be annotated explicitly, but the default behavior of Entity Framework Core is to implicitly infer the primary key from the C# property/field names. If a property in a entity class contains **Id** in its name, then Entity Framework Core elects that column as the primary key of that table unless specified otherwise. For specifying table relationships, **class composition** is used by either referencing a property with the type of the entity class by which that entity has a relationship to, or by using foreign key properties that reference the primary key of the related entity.

In the database context, it's where it's defined to Entity Framework Core what tables should be created (following the definitions from the C# entity classes in the Core Layer), what special rules should be applied to each table and columns (we can also explicitly define all the previously stated column configurations that can be made through annotations), what seed data should the database hold and what **table inheritance strategies** should be applied. A table inheritance strategy[10] defines how C# type hierarchy is mapped into the database, meaning how the base entity classes and specialized entity classes are treated in the database. There are three inheritance strategies available in Entity Framework Core: **table-per-hierarchy**, **table-per-type** and **table-per-concrete-type**.

In the developed work a table-per-type strategy was adopted on User, Patient and Physiotherapist tables. The User table holds the common fields that are shared between physiotherapists and patients, while the specialized tables Physiotherapist and Patient, hold the specific fields to each user type. The trade-off of this approach is that, it's a bit less performant and requires two entries (one on the User's table and another on the specialized table) but in turn doesn't pollute the User's table with unnecessary columns that only concern a specific user type. And by having this segregation into two tables, it is possible to specify more fine grained relationships.

After providing the corresponding table definitions in the database context, we can then leverage Entity Framework's Core Design tools, specifically the **`Add-Migration`** command, which basically maps our table definitions into C# instructions that generate raw SQL statements. After generating the database migration files, we run the **`Update-Database`** command which basically acts upon these generated files and translates the C# instructions to the respective SQL instructions that get executed on the selected database engine.

[10] EFCore Inheritance - https://learn.microsoft.com/en-us/ef/core/modeling/inheritance

In the developed work, since we are using a PostgreSQL database driver, the underlying C# instructions were translated into PostgreSQL statements in order to generate and populate the database.

After going through Entity Framework Core inner mechanisms, the database modeling will now be discussed.

A clinic typically comprises a set of physiotherapists, patients and its operational tasks or necessities. In the *KNEEMOR* system, the **Clinic** table acts as the root database entity of the system, i.e., it is the main entity where users belong to, meaning system state always belongs to a specific Clinic and is not shared between Clinics. A Clinic can have one or more Users (that can be either physiotherapists or patients) and a user must always belong to a Clinic. The Clinic table comprises the following columns: a name, address, vatnumber, phone number and a imageUri (in case during Clinic creation an image is provided). Clinics can only be created by users that hold the Admin role.

The **User** table as discussed previously, holds common fields that both physiotherapists and patients share (like name, email and address) and is also where authentication data is stored, specifically the user credentials and the refresh token used for refreshing the API's access token. Additionally the User table forms a relationship with the **UserRole** table and transitively with the **Role** table. These tables are used for authorization purposes for limiting access to users depending on the API routes that are accessed. The Role table holds the Physiotherapist, Patient and Admin roles respectively. The User's table also is anonymized in most its columns through the **EncryptColumn** annotation from EntityFrameWork.Core.EncryptColumn library. By anonymizing the patient's and physiotherapist's user data we are trying to comply with the data privacy guideline researched in the state of the art, allowing the information to only be in clear text if requested through the backend API.

A **User** can have one or more roles, but in the *KNEEMOR* system, upon registering as either a physiotherapist or a patient, only the specified user type role is assigned.

A Physiotherapist and Patient can have zero or more **Appointments**, **Protocols** and **Chats**, but an Appointment, Protocol and a Chat, must always have a Patient and Physiotherapist assigned to.

Each Physiotherapist also has seven **Availabilities**, corresponding to the seven days of the week. Each Availability row is comprised by the following columns: DayOfWeek (which specifies what week day that row corresponds to), StartTime (the time by which the physiotherapist becomes available for appointments), EndTime (the time by which the physiotherapist is no longer available for appointments) and isEnabled (which states if the physiotherapist works in that day of the week or not). Parallel to this the Physiotherapist can also specify zero or more **Unavailabilities**. A Unavailability comprises a **StartDate and EndDate** columns which indicate the unavailability period in which a physiotherapist cannot provide appointments.

In the *KNEEMOR* system, it was decided that each Appointment corresponds to a one hour time slot. Based on these two previous tables, if availability slots were stored for each

day, the number of rows to store in the database would be proportional to the available appointment time slots inferred from the 1 hour intervals from StartTime and EndTime boundaries, while also accounting the unavailability periods.

To avoid having to store these daily appointment availability time slots for all possible dates and also due to the fact that availabilities can be changed and adapted by the physiotherapist as he fits necessary, instead the **AppointmentAvailabilityView** was created for addressing this matter and correlating the two previous tables into appointment time slot availabilities by date period, by physiotherapist and by availability status (available, booked or unavailable).

The view has four input parameters, a start date, end date, a physiotherapist ID, and an availability status and returns a table with a date, day of the week, start time, end time and slot status.

To generate the availability days and time slots on the fly, a feature from PostgreSQL was used which is the **`generate_series`** set returning function.

This feature allows us to generate series of values between a contained boundary. In the view, this function is used to generate the various days between the input start and end dates, and for each generated date, this function is also used to generate the hourly time slots between 07h00 and 22h00. This correlation is possible through a cross join that combines each generated date with the pair of generated start and end times. After this generation, the availability itself for a given physiotherapist is then verified by checking the date and time slot information against the Availability, Appointment and Unavailability tables. The availability table checks if the physiotherapist is available in that time period and day of the week, the appointment rules out any existing appointment that might clash with the generated series, and the unavailability table rules out from the view result all the days by which the physiotherapist is unavailable. Through this view, it is possible to dynamically generate a physiotherapist's availability schedule for a given period of dates without having to store said information in the database.

An **Appointment** holds an appointment date, an appointment start and end times (which disclose the time slot for that appointment), a physiotherapistId, a patientId, a location (remote, home or clinic), a description and zero or more appointment reports (or **AppointmentData**). An AppointmentData comprises a summary, a diagnostic, a treatment and recommendations.

A **Chat** has zero or more **ChatMessages** and must always have one patient and one physiotherapist comprising it. Only Chats between physiotherapists and patients are available in the *KNEEMOR* system, this is due to a design decision of how the chat creation works. A chat creation is triggered when a pending appointment from a patient is accepted by a physiotherapist. Upon the physiotherapist accepting an appointment, the backend validates against the database if a chat already exists between the physiotherapist and the patient that required the appointment, if the chat does not exist then a new Chat is created and both parties can start interacting with one another. Due to this reason, only one Chat is available between a physiotherapist and patient. A ChatMessage comprises

the following columns: a SendTime, a chatId, a receiverId (user receiving the message), a senderId (user that sent the message) and isRead (to indicate if the message has been read).

A **Protocol** describes a group of one or more physiotherapy **Exercises**. It contains a description, a start and end dates and one or more exercises. A protocol must always have at least one physiotherapy exercise assign to it and an exercise must always belong to a protocol.

An **Exercise** is a set of parameterizations that are specified for a given **ExerciseType**. These parameterizations encompass minimum and maximum angles, number of reps and sets, rest and hold times (for stretching exercises) and also which knee should be employed in the exercise. The **ExerciseType** holds presentation metadata related to an **Exercise**, it provides a name and also a image and video URIs for visualization purposes for that exercise. Also an **Exercise** has one or more **ExerciseSchedules**. An **ExerciseSchedule** consists on a day of week, start and end times and a column to identify if that day of week is enabled or not.

Still related to **Exercises**, we have **ExerciseCompletions** which dictate if a given **Exercise** was performed for a given date and additionally also holds one or more **ExerciseReadings**. The **ExerciseReadings** correspond to the motion data that is recorded from wearable sensor whilst undergoing a physiotherapy exercise.

The final entity relationship diagram is depicted in Figure 4.5 for the reader's better comprehension and visualization of the overall entity relationships.

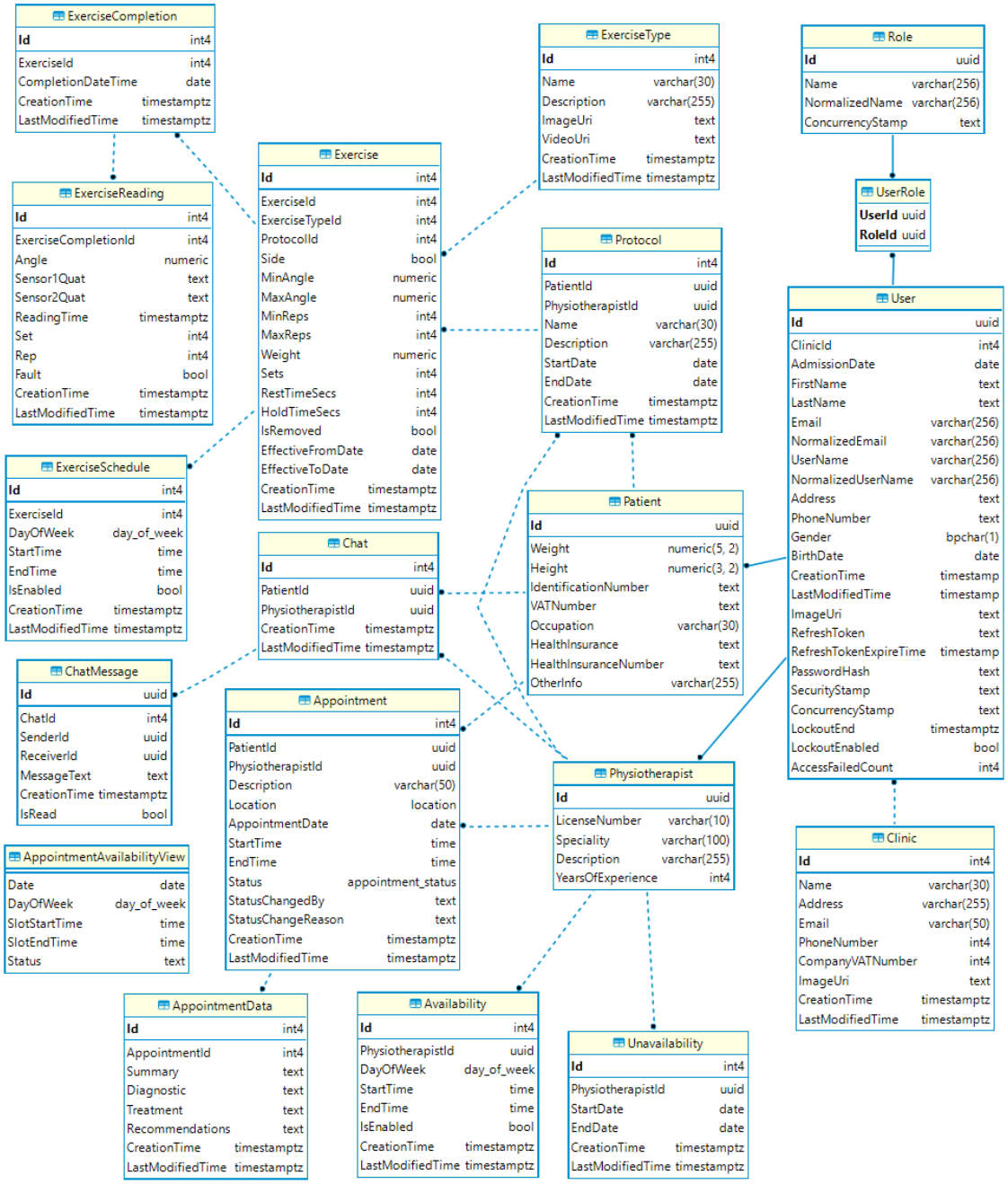


Figure 4.5: KNEEMOR Database Diagram

4.2.3 Backend API design and development

The backend API is the point of entry of the backend layer. It is responsible for authenticating and authorizing incoming requests and also provide a medium of communication between the frontend applications and the respective backend resources (database and cloud storage).

The backend API was built using ASP.NET Core (.NET 7 version) following REST API standards, meaning all the endpoints are stateless, are correctly identified using plural nouns, their related resources are structured hierarchically and also use the proper HTTP methods depending on the operations that are performed over a resource.

In the Web layer project, the backend API is structured into endpoint group classes, where each group class holds one or more endpoints. Each endpoint group is prefixed with the API version (/api/v1), followed by a prefix related to the endpoint route group.

The endpoint groups that are defined in the backend API are the following: **ClinicEndpoints, UserEndpoints, PatientEndpoints, PhysiotherapistEndpoints and ExerciseTypeEndpoints.**

The **ClinicEndpoints (/clinics)** group holds all endpoints related to operations over a Clinic entity. This endpoint is used for retrieving a Clinic's identification details as well as perform updates over those details in clinics.

The **UserEndpoints** group provides the registration and login endpoints, and also provides the refresh token endpoint which is responsible for reissuing a new access token when the current token has expired its lifetime.

The **PatientEndpoints (/patients)** group holds all the capabilities a patient has for interacting with the *KNEEMOR*'s system data. It provides endpoints for viewing the current weekly exercise completion progress, viewing and booking appointments, view appointment's data, view assigned protocols and exercises, register exercise readings, view chats and also export reports related to appointment data and exercise readings.

PhysiotherapistEndpoints (/physiotherapists) group on the other hand, holds all the capabilities a physiotherapist can perform in the *KNEEMOR* system's data. The endpoints offer the physiotherapists with the capabilities of viewing upcoming and pending appointments, viewing and managing appointments, managing patient information, managing appointment availabilities and unavailabilities, viewing and managing protocols, exercises and also exercise schedules, view their corresponding chats and also export reports related to appointment data and exercise readings.

Lastly, the **ExerciseTypeEndpoints (/exercise-types)** group covers the endpoints for allowing physiotherapists to perform the creation/update of exercises types. These hold a title, description and optionally corresponding multimedia content (image, video or both).

In each endpoint, **OpenAPI** annotations were used with the **NSwag** library, to not only provide a API documentation reference in form of a web page but also provide the capability of generating API clients with ease. For generating the Typescript Axios

API client used in both frontend applications, **NSwagStudio** was used to consume the underlying OpenAPI specification and generate the desired Typescript Axios client. By generating the API client automatically from the OpenAPI specification, this saved a lot of development time on the frontend applications.

For interacting with the database entities, a combination of **DTOs, DAOs and MediatR command and requests** were used. The DTOs are used for transporting frontend request data payloads and also are used for delivering the results mapped from data access objects back to the frontend applications. This segregation allows for database entity changes without directly affecting the frontend contracts and also promotes loose coupling between the Application and Core layers.

Supporting each endpoint request, depending if it is a query or mutation over the database, MediatR requests and commands are used.

These commands and requests are organized by features (or entities) in the backend's Application layer project, to promote separation of concerns and also to ensure that for each request, the necessary logic is defined in a well-isolated class. These commands and requests are forwarded from the Web layer endpoints onto the Application layer, where they are handled accordingly based on their handler classes.

These handler classes have the required services and repositories injected for fulfilling the underyling request, and after the required operations are concluded, a result DTO is returned back to the Web layer and the appropriate response is generated.

For storing the multimedia data for Clinics, Users and also ExerciseTypes, the **Azure BlobStorage** cloud service was used. On Azure, a Resource Group had to be first created, followed by a Storage account. During the creation of the storage account, the blob access tier had to be chosen between cold or hot tiers. Cold tier is typically used for blob data that is infrequently accessed, and also requesting said data is typically more expensive. The hot tier instead allows for frequent access to blob resources, typically blob data that gets requested frequently and needs quick access. This tier is cheaper compared to the Cold tier since in both frontend applications, we need to retrieve the profile images and also the exercise multimedia. For storing said multimedia data, after the Storage account was created, there was also the creation of three blob containers respectively, **clinic, exercise and profile** containers.

When creating either a **Clinic, ExerciseType or a User** in the respective frontend applications, a prompt is displayed to the User if he'd like to add multimedia content. In the case of a Clinic and a User only an image may be submitted and in the case of an ExerciseType an image and a video may be submitted. Upon selecting the desired media, there's file size and extension validation in the frontend and in the backend, and additionally in the backend there is also a file extension signature validation which verifies if the submitted file has the prefix bytes related to a specific file extension. If all validations on the files pass, then the file gets uploaded to the respective blob container and uses the following convention for the file name, `{entity-name}-{id}-{image|video}.{file-extension}`.

In case of profile images, we can update the underlying profile image of physiotherapists but for patients this capability was disabled to prevent unnecessary uploads and cost increase of the blob storage service.

For exporting the exercise readings and also appointment data, some **report endpoints** were created to provide data exportation in different formats (**`.csv, .xlsx and .pdf`**). To build these reports, templates had to be created in each format and when requests arrive to these endpoints, the templates are used for producing these reports.

The reports created in the developed work include an **appointment data report** (holding a summary, diagnostic, treatment and medical recommendations), an **exercise completion report** (which comprises the readings from an exercise) and a **monthly protocol report** indicating the number of exercise completions by day of the week per exercise, and, for each exercise the monthly average of knee angle readings with a statistics table comprising the mean angle, minimum angle, maximum angle and number of completions for the month as depicted in Figure 4.6.

(a) (b)

Figure 4.6: Monthly Protocol Report

Other types of reports could've been created and there's the possibility for extension on the existing reports, but due to time constraints only the **`.pdf`** format is available for the exercise readings and monthly reports, while the appointment data report can be viewed in all three formats.

4.2.4 Authentication and Authorization

Depending on which endpoint a user accesses, authentication might be required in order to interact with the underlying resources.

For handling registration, authentication and authorization concerns, ASP.NET Core Identity was used. When a user registers in the *KNEEMOR* system, .NET Core Identity enforces password validation rules to ensure strong password strength. The rules that were defined were, at least one digit, one lowercase letter, one uppercase letter and one non-alphanumeric character to strengthen the password. In addition to this, the password is hashed using the default **`PasswordHasher<TUser>`** implementation provided from .NET Core Identity.

This implementation uses the cryptographic algorithm Password-Based Key Derivation Function 2 (PBKDF2), which performs hashing using the **HMAC-SHA256** hashing algorithm over a combination of a securely generated random salt value, the user's password and an iteration count[11]. The generated password hash contains the version of the `PasswordHasher`, the hashing algorithm, the iteration count, the salt size, the salt value and the derived secret key from using PBKDF2 over the password and salt. By using an high iteration count and a unique randomly generated salt, it makes the password hash more impervious to brute-force attacks and dictionary attacks. However using this algorithm impacts the backend's performance depending on the iteration count that is provided, the higher iteration count the more resource heavy authenticating requests become.

Once the registration process concludes, a **`TokenDetailsDTO`** is produced as result. This data transfer object holds a **JWT access token** containing some claims (clinicId, userId, username, roles) and also a **refresh token** which is used for refreshing the access token.

For generating the JWT access token, the hashing algorithm **HMAC-SHA512** is used with a 64 byte long key for signing and validating the JWTs, and also a token expiration is specified. The refresh token is generated using a cryptographically secure **`RandomNumberGenerator`** which then populates 64 byte long array that then gets converted into a base64 string. During the registration process the refresh token is stored in the User's table along with it its expiration time.

Each time a JWT access token becomes expired, the frontend applications send a refresh token request, which generates a new access token and also updates the existing refresh token and it's lifetime, before attempting the previous failed request.

Still regarding endpoints and endpoint groups, these are protected from unauthorized access by using **Authorization Policies**, which basically assert if an incoming request complies to the conditions/rules established for accessing underlying endpoint or endpoint group. The **Authorization policies** work in conjunction with the **Roles** table to provide the backend with role-based access control. There are six authorization policies: **admin,**

[11] Beware of the default ASP.NET Core Identity settings - https://www.strathweb.com/2023/03/beware-of-the-default-aspnet-identity-settings/

physiotherapist, patient, physiotherapist_common, register and refresh.

The **admin** policy checks, if the current user issuing the request, has the Admin role, and this policy protects the endpoints referring to creation and updates of clinics.

The **physiotherapist** policy asserts that the user has the Physiotherapist role and that he has ownership over the resource he accessed, by comparing the request path variable **`physiotherapistId`** against the access token claim **`userId`**, allowing to go through the request only if it is a match. This policy protects the **PhysiotherapistEndpoints group**.

The **patient** policy does the same as the physiotherapist policy but instead protects the **PatientEndpoints group**, and also if it is a physiotherapist accessing any patient endpoint, the path variable assertion is bypassed and the request is allowed to go through.

The **physiotherapist_common** policy asserts only if the user has the Physiotherapist role and protects the **ExerciseTypeEndpoints group**.

The **register** policy checks if the user has either the Physiotherapist or Patient roles, and also checks if the underlying path variable matches with the user that is performing the registration. This policy is used when creating the multimedia resources in the BlobStorage service from Azure.

Lastly the **refresh** policy checks for both Patient and Physiotherapist roles, and is used to protect the refresh token endpoint.

As said in previous chapters, some of the User table columns are **anonymized** using EntityFrameworkCore.EncryptColumn library. To achieve the anonymization, the **AES-128** symmetric-key encryption algorithm is used under the hood provide the encryption/decryption of the tables columns as queries are made through EntityFrameworkCore. While AES-128 encryption is considered safe, a stronger variant like AES-256 could've been used, as it offers more resiliency to brute force attacks compared to the 128 bit version but at a cost of performance in terms of encryption/decryption times [14]. However, due to the library not allowing changes on what encryption algorithm to use, AES-128 was instead used.

To store the backend application secret keys, i.e. the JWT secret key used for signing, the BlobStorage API key and also the key used for the symmetric encryption provided by the EncryptColumn library, theses secret values were stored in the **User Secrets**, which is a .NET Core feature for storing sensitive data locally. In the applications code we instead use setting variables that read from this local storage the underlying secrets instead of hard coding these values in a normal configuration file or in code.

4.2.5 Notifications Hub

The Notifications SignalR Hub is the component responsible for ensuring the Chat communication between the patient and physiotherapist applications.

When either the physiotherapist or a patient is online in either frontend application, an attempt is made for establishing a connection with the Notifications SignalR hub. This connection attempt requires passing the access token as a query parameter, so that in the

backend application when it receives a Websocket connection attempt, it is able to extract the access token and append it to the SignalR's hub context.

Inside the Notification's Hub, the **`OnConnectedAsync`** handler is where the connection request is handled and where we are able to access the hub's context and check if the access token is present. If the access token is not present, the connection is rejected, otherwise, we extract the **`userId`** claim from the JWT access token claims and store it as the key in a thread safe dictionary along with its value, the **`connectionId`** that gets automatically generated when performing a new Websocket connection. By storing this key-value pair, we are able to correlate connections with users while ensuring there isn't concurrency issues when accessing and storing information in this dictionary.

When ChatMessages are sent from either application, a **`CreateChatMessageDTO`** is created from the frontend applications and is sent to the respective handler in the Notifications hub. After passing the `CreateChatMessageDTO` into MediatR's respective command handler and after the message is stored in the **`ChatMessage`** table, a verification is made against the thread safe dictionary to check if there is any user connected currently which is equal to the **`receiverId`** field comprised in the `CreateChatMessageDTO`. If there is a user matching this criteria, then the `ChatMessage` is forwarded to the respective user by using his **`connectionId`**, and the respective patient or physiotherapist receives the message in real-time.

When internet connection is lost or a user logs off, the **`OnDisconnectedAsync`** handler kicks in and removes the disconnected user from the connected users dictionary.

4.3 Deployment

During early development stages, the backend server, mobile app and the progressive web app would run separately on their own development ports. As consequence of this, on the backend server cross-origin resource sharing (CORS) policies had to be placed in order to allow HTTP requests to flow through the backend server from both client applications[12].

Despite being possible to deploy the applications separately, it's not ideal nor practical in a deployment point of view, since it requires users to install manually all the needed dependencies from all applications, ensure proper installation and configuration of all services and also follow all the required steps by order in terms of what's required for the deployment of the system.

Due to these various points, a better approach was explored which was containerizing the whole backend layer with the physiotherapist's progressive web app sharing the same container.

By leveraging a containerization platform, like **Docker**, it enables packaging of applications into images and run these images in containers on whatever operating system platform we target our applications.

[12]Cross-Origin Resource Sharing - `https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS`

This relieves the need of managing dependencies on different operating systems and provides an easy deployment option for applications, be it in production or for local test purposes.

For containerizing the backend application, a **`dockerfile`** was created through Microsoft's Visual Studio **`"add container support"`** functionality, which basically generates the needed `dockerfile` instructions for providing containerization of the application.

Despite Visual Studio providing a generated `dockerfile`, some changes had to be made in order to apply the migration files that were required for the backend, specifically seed data that was used for testing porpuses.

To solve this issue, it was additionally added to the container the Entity Framework Tools which provide the needed commands for creating and applying migrations to target database.

Other deployment options were also available like using **Azure App Service** or even performing a deployment setup on a virtual private server. However due to time constraints and also due to the wide scope of the thesis, the *KNEEMOR* system was only tested locally and never got the chance to be tested in a testing environment using one of the mentioned deployment options.

4.4 Frontend Layer - Physiotherapist Application

The physiotherapist application is the *KNEEMOR* system component that enables physiotherapists to manage various aspects of patient care and rehabilitation. It allows them to manage patient profiles, assess knee range of motion through visual graphs, and also export medical reports. It also provides physiotherapists with the ability to create and assign rehabilitation protocols, which are comprised of exercises tailored to each patient's specific needs and motor capabilities. Additionally, physiotherapists can approve and manage appointments, set up their appointment availabilities and unavailabilities, and also the ability to create appointment data that includes a detailed diagnostic and recommendations for patients. Lastly, the application includes a Chat system, enabling physiotherapists to exchange messages with their patients.

4.4.1 Project structure and authentication flow

The physiotherapist progressive web app was developed using **React and Typescript with Vite.js**. By using Vite.js [13] for building our physiotherapist application, we can benefit from fast hot module replacement and also from a panoply of available plugins Vite provides for easing development. One of these plugins was crucial for providing PWA with minimal setup. For rapid prototyping and styling **TailwindCSS** was used, due to offering quick CSS classes that allow for more customized styles compared to component-based libraries and also since to the familiarity to the framework.

[13] Vite.js - Next Generation Frontend Tooling - https://vitejs.dev/

Based on the requirements elicited in the proposal chapter some Javascript libraries were used to accelerate the overall development: **SignalR Client** (for interacting with the Notifications Hub from the backend), **Redux Toolkit** (global state manager for storing the state application state), **TanStack React Table** (a robust table component that provides various capabilities), **Axios** (an HTTP client for issuing HTTP requests), **react-chartjs-2** (a simple chart library based on Chart.js), **react-i18next** (internationalization library for translation purposes), **jwt-decode** (a library for decoding JWT payloads), **lottie-react** (a library for providing JSON based animations), **moment.js** (library for handling date and time manipulation), **react-hook-form** (a library for handling form management), **react-router-dom** (for providing routing), **redux-persist** (for persisting redux state in local storage) and **vite-plugin-pwa** (for providing pwa capabilities for the react application).

The physiotherapist application is structured in four main folders, **assets** (for storing icons and images), **components** (for storing auxiliary react components), **pages** (which represent each page or functionality a physiotherapist has access to) and **services** (which holds utility functions and hooks for interacting with different services, like the backend Axios API client, SignalR client, i18n translations, notifications and network status hooks, and lastly the Redux store and its various state slices). Also the application supports two languages, Portuguese and English, by using the i18n React library and the corresponding translation files.

A physiotherapist before interacting with the application must first login or register himself. The registration process requires the physiotherapist to choose a Clinic to be assigned to and fill the required form data to conclude the registration. Form validation is provided through react-hook-form, to check if all data has been filled before accepting the registration from the physiotherapist. After the physiotherapist submits the form and his account is created in the backend, an access token comprising his `clinicId, userId, username and roles` is issued, along with a refresh token. Theses claims are stored and persisted in a Redux state slice which can be accessed through the application by using its state selector hook. After receiving both tokens, the physiotherapist is taken to an additional form to select his profile image. The physiotherapist can opt-out from this step and have no profile image for the time being.

When logging in to the application or accessing any page that requires authentication, the existence of the access token is verified and also the user roles are checked to see if they have the required role to access the specific page.

This verification is ensured through a **`ProtectedRoute`** component that is wrapped around one or more react-router-dom routes. Through this component we specify what roles the underlying routes require, and in case the user is unauthorized or his access token is missing, then the user gets redirected to an error page where a 401 status code is displayed.

When performing API requests, if the access token becomes expired, a Axios response interceptor is in place to handle the token expiration. This middleware/interceptor sends the current access token along with the refresh token to the backend so they can be

refreshed. After receiving the refreshed tokens, the initial API request that failed due to token expiration is then reissued and the frontend applications continue their work.

4.4.2 Appointment and Availability management

One of the main daily operations a physiotherapist has to perform is manage their **Appointments** and **Availabilities**. The physiotherapist application was developed with those concerns in mind. When a new physiotherapist is created in the *KNEEMOR* system, his appointment availabilities are also created transitively with some default time periods and with all the days of the week disabled.

The physiotherapist upon logging in to the application, must then change their availability accordingly and enable the days of the week that he intends to do his practice as seen in Figure 4.7a. Also, in case a physiotherapist has to be unavailable for a period of days, he can specify a unavailability date period by which he does not provide appointments.

The physiotherapist, by setting the availabilites and unavailabilities for his appointments, directly impacts and reflects on his appointment schedule booking for patients, since the view that is used to retrieve the appointment availability slots uses the information from these two tables and also uses the information provided from the appointments table.

When a patient performs an appointment booking, the booking is created in the Appointments table, and its appointment state is set to **`pending`**. The physiotherapist then can decide if he approves or rejects the appointment. If the appointment gets **`Accepted`**, a **ChatMessage** is created in the backend and is sent to the patient confirming the appointment approval.

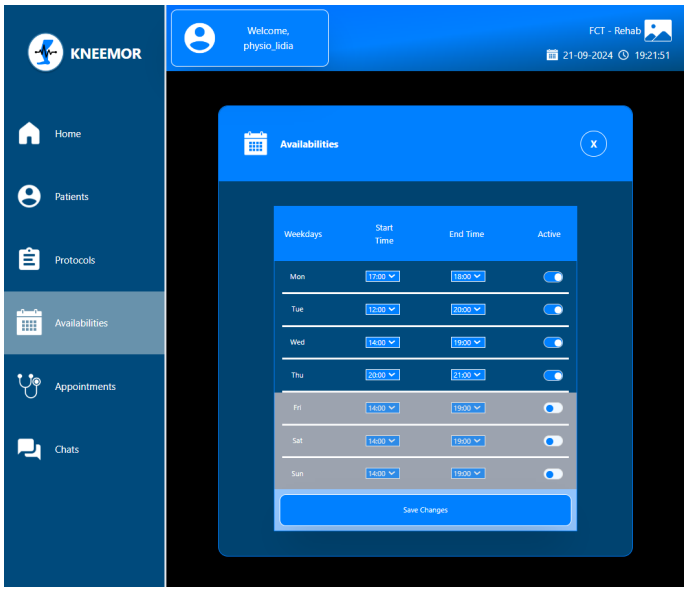
After appointment being created, the physiotherapist can also attach **AppointmentData** to a specific appointment.

The **AppointmentData** holds a summary, a diagnostic, a treatment and possible recommendations that the physiotherapist might provide in his appointment as seen in Figure. This appointment data can then be exported by both the patient and the physiotherapist in different file formats (**`.csv, .xlsx, or .pdf`**) as depicted in Figure 4.7b.

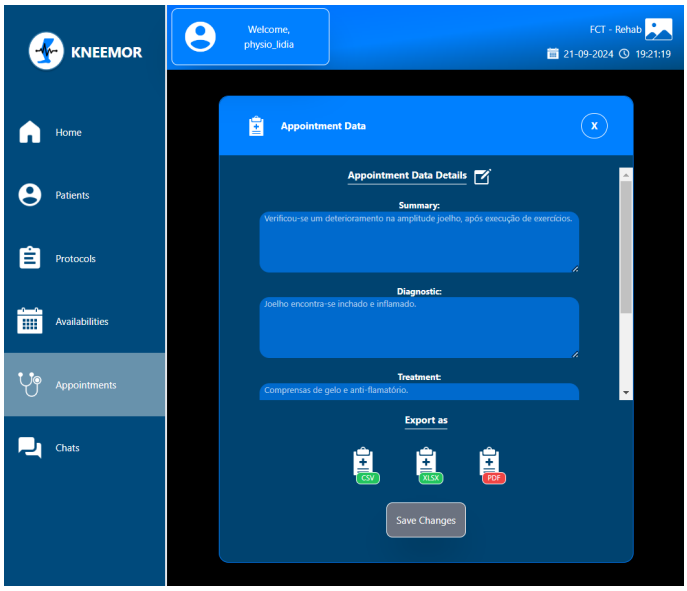
4.4.3 Patient and Protocol management

The physiotherapist application besides providing the capability of managing appointments, has the main functionality of providing the **`Protocol`** and **`Exercise`** creation and assignment to patients.

Before a physiotherapist is able to create any protocol, at least one **ExerciseType** must first exist in the database. An ExerciseType describes what type of exercise must be performed. It contains a name, a description and optionally, a image and a video as seen in Figure 4.8. When an ExerciseType is created in the backend application, it returns the **ExerciseTypeId** to the physiotherapist application, so it can be assigned to the filename



(a)



(b)

Figure 4.7: (a) Availabilities and (b) AppointmentData pages

that will be used in Azure Blob Storage. The selected multimedia content then gets uploaded into the exercise blob container, and after the upload finishes, an `imageUri` or `videoUri` is returned to the backend, and the respective ExerciseType gets updated in these fields.

After at least one ExerciseType is created, the Protocol creation can then begin. As discussed in other sections, a rehabilitation Protocol consists of one or more Exercises, where each Exercise has an associated ExerciseType, one or more ExerciseSchedules and a set of exercise parameters.

These exercise parameters encompass the minimum/maximum angles by which the exercise is valid, the minimum/maximum repetitions, the number of sets, which knee is performing the exercise (left or right) and optionally the rest and hold seconds. The hold seconds serve for adjusting the validation time over the target angle and indirectly can be viewed as stretching exercises.

By having these exercise parameters, a physiotherapist can create different types of exercises that are adequate to the motor capabilities of the patients and adjust them accordingly.

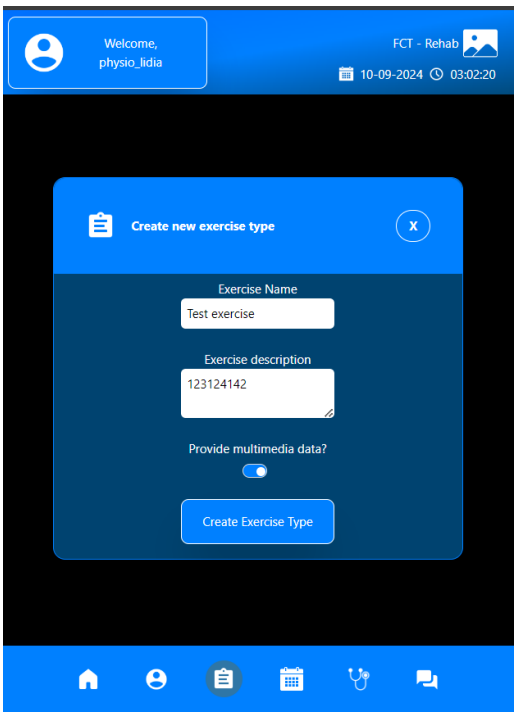
When a Protocol has been created, the underlying exercises can be modified and new ones can be added. Also when an Exercise is updated, a soft delete happens in the Exercise table, meaning the exercise is not deleted and instead is marked as removed, and its EffectiveToDate is filled. The newly updated exercise is then added to the Exercise table, with the reference of `Exercise` that was removed, this way it is possible to view the parameters that were active in a past date.

After the patient performs exercise sessions, ExerciseCompletions and ExerciseReadings are generated. The physiotherapist can then view the ExerciseCompletions that are available in that protocol in a report manner or can also get a monthly overview of said protocol and assess on how patient is doing in terms of range of motion per exercise and also in terms of completing the exercise schedules.

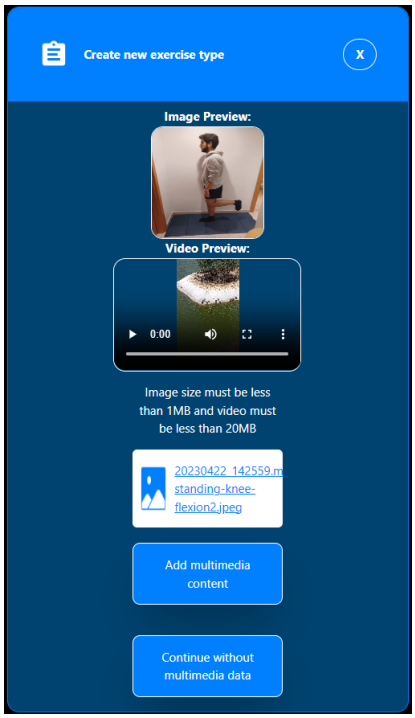
4.4.4 Chat system

When an appointment is accepted from the physiotherapist, it triggers a Chat creation on the backend only if it is the first appointment with the respective patient. From that point forward, chat message communication between that physiotherapist and that specific patient can happen. When accessing the Chat list page, the profile picture of the patient, his last message and the last message time is displayed to the physiotherapist. In case the physiotherapist application loses internet connection, the underlying chat details can't be accessed. Upon entering a specific chat, the chat messages are ordered descended by CreationTime, so that the physiotherapist can view from the newest to the oldest chat messages first as seen in Figure.

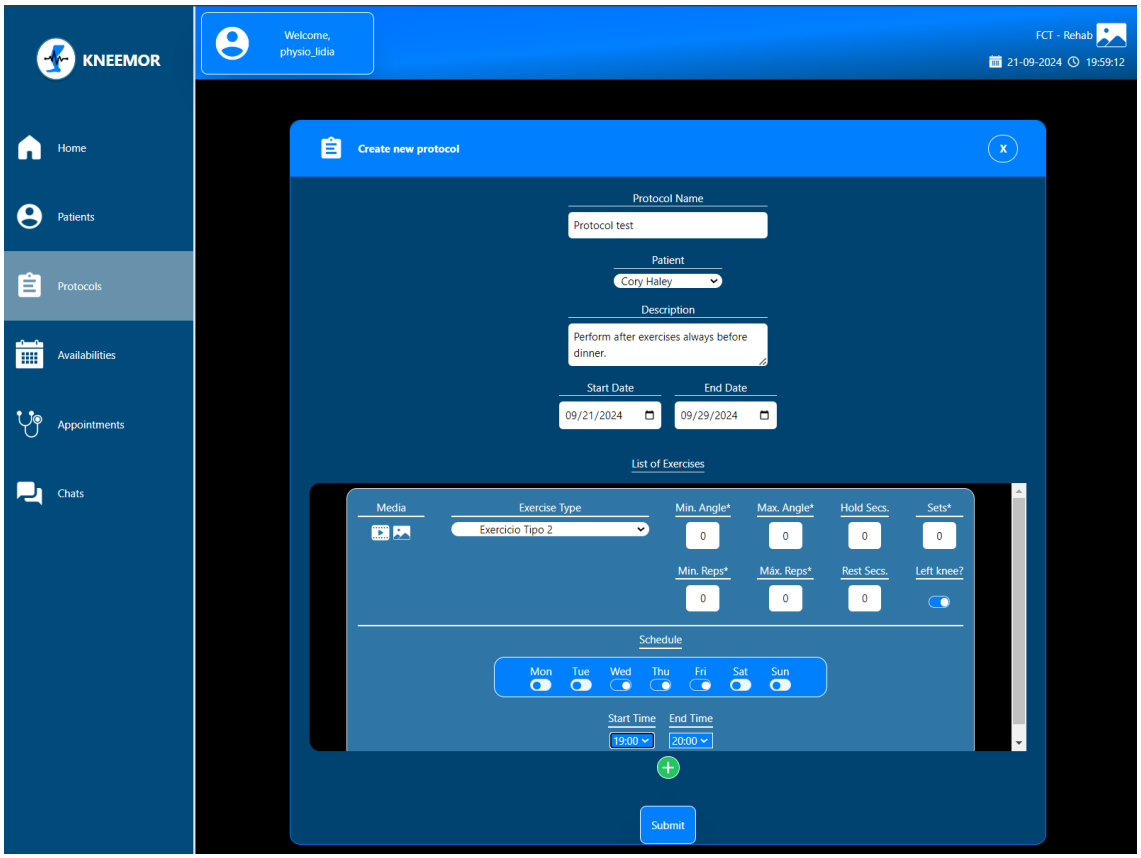
When a chat message is sent to the patient, the SignalR client invokes the **`SendChatMessage`** method from the Notifications Hub on the backend. The method in turn stores the chat



(a)



(b)



(c)

Figure 4.8: ExerciseType and Protocol creation.
(a) ExerciseType details, (b) ExerciseType media, (c) Protocol creation

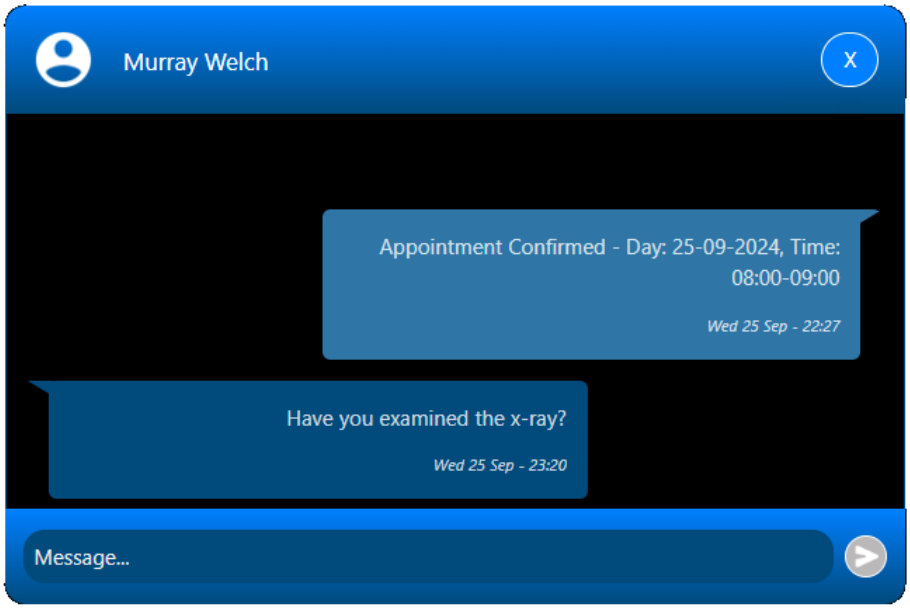


Figure 4.9: Chat between a physiotherapist and a patient

message in the database, and then relays the message to the patient's application by invoking the **`ReceiveChatMessage`** event declared on the patient's SignalR client. When the message arrives in the patient application a notification is triggered and the process repeats itself when a patient sends a message. Also an important note is that the chat messages are immutable, meaning once they are sent, they cannot be updated. This could be improved in future work and also additional features could be added to the chat.

4.4.5 Offline support

The physiotherapist application provides some degree of offline support due to its PWA capabilities and also thanks to redux-persist for persisting application state in local storage and a react context which checks for internet connectivity.

The Vite's PWA plugin provides the physiotherapist application with a service worker that gets dynamically created. This service worker is able to intercept incoming network requests and cache them accordingly based on caching strategies that are specified in Vite's configuration file.

In the configuration file we can define **prechaching** and also **runtime caching**. The **prechaching** is configured through the **`globPatterns`** property which defines what files should be cached at the service worker's build time[14]. This means when the dynamically generated service worker is created, the specified files are already bundled into the service worker not requiring it to cache later. The **runtime caching** as the name implies, caches assets as they are requested through the web application's runtime. Based on a cache strategy and a `urlPattern` property, we can define what API requests or static assets we

[14]VitePWA - Service worker precache - https://vite-pwa-org.netlify.app/guide/service-worker-precache

want to cache and also specify the cache lifetime so that the cached assets can turn stale and be re-fetched again.

In the physiotherapist application, the profile and exercise images stored in Azure Blob Storage get cached using a **CacheFirst** strategy, which basically checks the cache for the image first and if it's not available in the cache, it then proceeds with fetching the image from specified imageUri. For videos the CacheFirst strategy didn't work properly, and instead a **NetworkFirst** strategy was adopted, which fetches the video first from the videoUri and, if it can't, then checks the cache.

To provide some degree of offline support and network status visualization, a **`NetworkStatus`** react context and redux-persist was used. The `NetworkStatus` context every 5 seconds performs a heartbeat request to the backend API to check if there's still internet connectivity available. If the `NetworkStatus` context detects that internet connection has been lost, then the `isOnline` state variable present in the `NetworkStatus` context is set to false, and a warning banner is displayed on the physiotherapist application as seen in Figure 4.10.

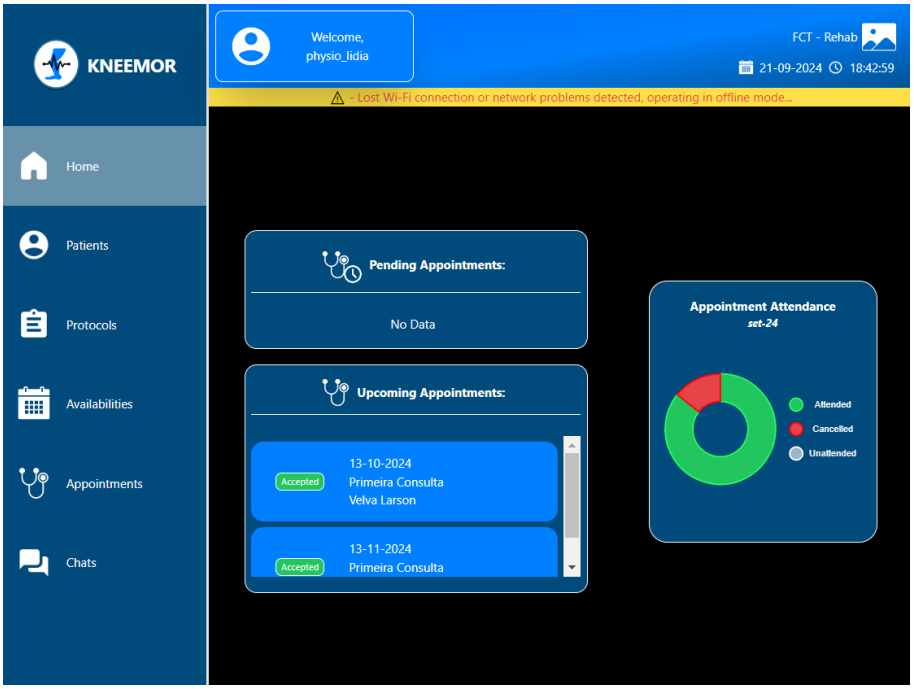


Figure 4.10: Offline mode banner

When in offline mode, the application then only relies on the state that redux-persist has stored until that point, and does not issue any more requests to the backend. The same behavior happens in the patient application.

Despite these offline capabilities being available in the physiotherapist application, the application was tailored for having a internet connection, meaning the offline support could be improved in the future.

4.5 Frontend Layer - Patient Application

The patient application is the *KNEEMOR* component which directly interacts with the Wearable Sensor Prototype. It is through this application that patients undergo the assigned physiotherapy exercises and where they view their overall knee ROM. A patient can view his exercise completion weekly progress, book and view appointments, view protocols and exercises, perform exercises and also view the mentioned reports that physiotherapists also have access to. Like the physiotherapists, they can also chat with physiotherapists that are accompanying them.

4.5.1 Project structure and authentication flow

The patient application was developed using **React Native, Typescript and Nativewind (TailwindCSS but in React Native)** due to providing development synergy with the physiotherapist application due to its similar codebase styles.

To fulfill the requirements that were enlisted in the proposal chapter the following libraries were used to simplify and accelerate the development of the application: **SignalR Client** (for interacting with the Notifications Hub from the backend), **Redux Toolkit** (global state manager for storing the state application state), **Axios** (an HTTP client for issuing HTTP requests), **react-native-gifted-charts** (a chart library using react-native-svg), **react-i18next** (internationalization library for translation purposes), **jwt-decode** (a library for decoding JWT payloads), **lottie-react-native** (a library for providing JSON based animations), **moment.js** (library for handling date and time manipulation), **react-hook-form** (a library for handling form management), **react-navigation** (for providing navigation between screens), **redux-persist** (for persisting redux state in local storage), **react-native-encrypted-storage** (a library to provide secure storage for secrets), **netinfo** (for providing an API to check the device's network status), **react-native-blob-util** (a library for facilitating file access and data transfer), **react-native-ble-manager** (a library for interacting with the native bluetooth module), **react-native-sound** (a library for playing sound effects), **react-native-video** (a library for providing a video player), **notifee** (a library for handling notifications) and other libraries for building some UI components.

The patient application is structured into six folders, **assets** (for storing icons and images), **components** (for storing auxiliary react native components), **navigation** (for storing all the navigator components which structure the navigation of the application), **screens** (for holding all the application screens and views) and **services** (which holds utility functions and hooks for interacting with different services, like the backend Axios API client, SignalR client, i18n translations, react native hooks, and lastly the Redux store and it's various state slices). Like the physiotherapist application, it also supports two languages, Portuguese and English.

A patient before interacting with the application must first login or register himself. The registration process is similar to the physiotherapist application but with different

fields related to the patient's perspective. After a patient authenticates himself, the access token along with the refresh token is stored in the Authentication slice from the Redux store, but in the patient's application, the Redux store instead of being persisted in local storage like the physiotherapist's application, is instead saved in the mobile device using **EncryptedStorage**, which stores the Redux state in cryptographically secure keystore via **EncryptedSharedPreferences** provided by Android[15]. By using the EncryptedStorage on the patient's application we ensure another layer of security in the *KNEEMOR* system by providing data confidentiality for the patient's data.

For navigating around the patient application, the access token is also validated and the same refresh token logic is in place like the physiotherapist application. The main differences rely on how navigation works on Android compared to typical web pages.

In the developed work for providing navigation between different screens, two types of react-navigation navigators were used: a **`StackNavigator`** and a **`BottomTabsNavigator`**.

The **`StackNavigator`** works like a stack, meaning as the user navigates between screens, each new screen is pushed onto the top of the stack of screens. When a user presses the back button, the current screen that is at the top of the stack is popped, and the screen that was next in the stack pile is then used.

The **`BottomTabsNavigator`** instead of a stack, uses a bottom tab component to navigate between different tab screens.

With these two navigators, the patient application navigation was structured into five navigators, each holding a set of screens and navigators to map out all the possible screen routes.

The application starts at the **`AppStackNavigator`** which nests two other navigators, a **`AuthStackNavigator`** (which holds the login and register screens) and a **`MainBottomTabs Navigator`** (which holds the various tabs pertaining to each feature of the patient application) as seen in Figure 4.11.

If the user is not authenticated, the `AuthStackNavigator` is presented, otherwise the `MainBottomTabsNavigator` is presented instead.

In each tab from the `MainBottomTabsNavigator` there's also nested `StackNavigators` for each feature related to Appointments, Protocols and Chats.

By grouping the related screens in each `StackNavigator`, user navigation and experience is facilitated and improved.

In the first time a patient enters the application, he is greeted with a brief tutorial explaining how the application works and requests for the various permissions that are required throughout the application's use, i.e. bluetooth, storage and also notification permissions.

[15]EncryptedSharedPreferences - https://developer.android.com/reference/androidx/security/crypto/EncryptedSharedPreferences

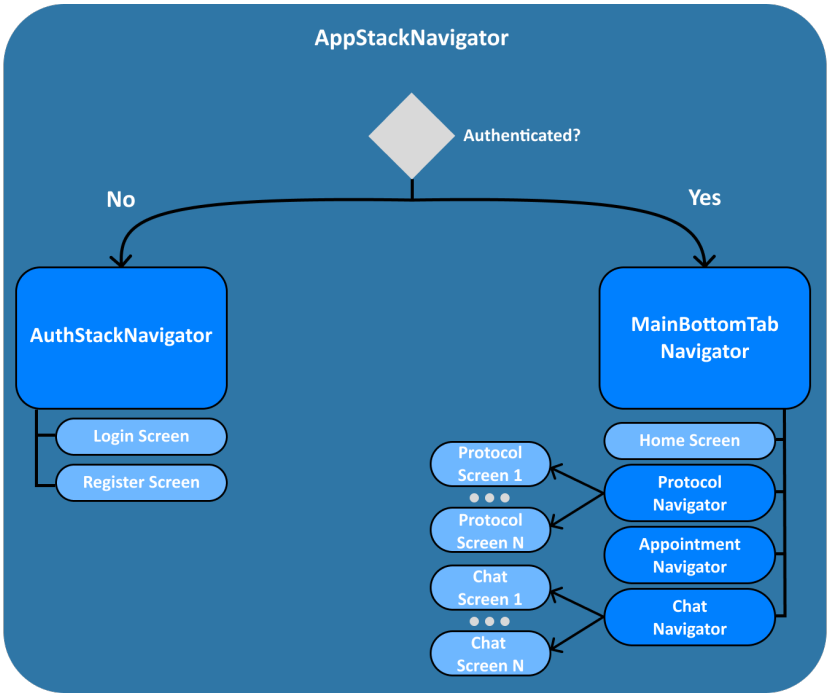


Figure 4.11: Patient Application Navigation Structure

4.5.2 Appointment Viewing and Booking

In the `Appointments` tab is where the patient can view his past, current and future appointments, book new appointments and also view appointment data that is attached to a particular appointment.

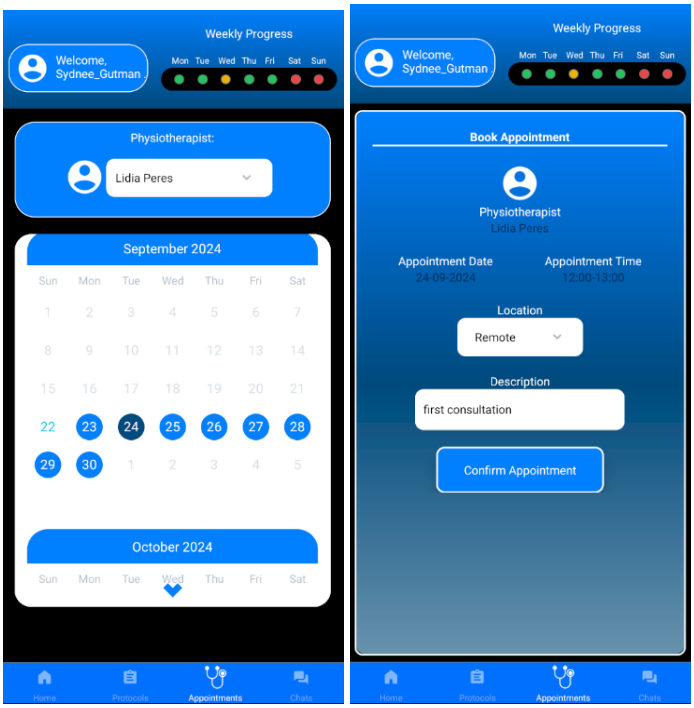
When the patient accesses the **`BookAppointmentScreen`**, he can choose which physiotherapist to book an appointment with and visualize the physiotherapist's profile image (if it has one).

By selecting a physiotherapist from the dropdown component, the underlying appointment availability slots are loaded for that specific physiotherapist in a calendar component, where the patient can view which days and time slots are still available for booking as seen in Figure 4.12.

The patient, after selecting the desired appointment time slot, is presented with a confirmation screen, so that the patient can fill the location of the appointment as well as the description or reason for the appointment booking. After confirming the appointment, he is taken into the created appointment screen, where he can view the current status and the underlying data that is associated with the appointment.

Additionally, in the **`AppointmentScreen`** he can also download his underlying appointment data (if they exist), in the formats that were mentioned in previous sections.

For using export report functionality, react-native-blob-util was used and some android permissions were required, specifically **`READ_EXTERNAL_STORAGE`**, **`WRITE_EXTERNAL_STORAGE`**, **`DOWNLOAD_WITHOUT_NOTIFICATION`** permissions. Under the hood, react-native-blob-util uses Android's Download Manager API for providing the download progress



(a) (b)

Figure 4.12: Appointment Booking Screens
(a) Appointment Booking Calendar, (b) Appointment Confirmation

and also for storing the report in the APK folder related to the application.

4.5.3 Protocol and Exercise Viewing

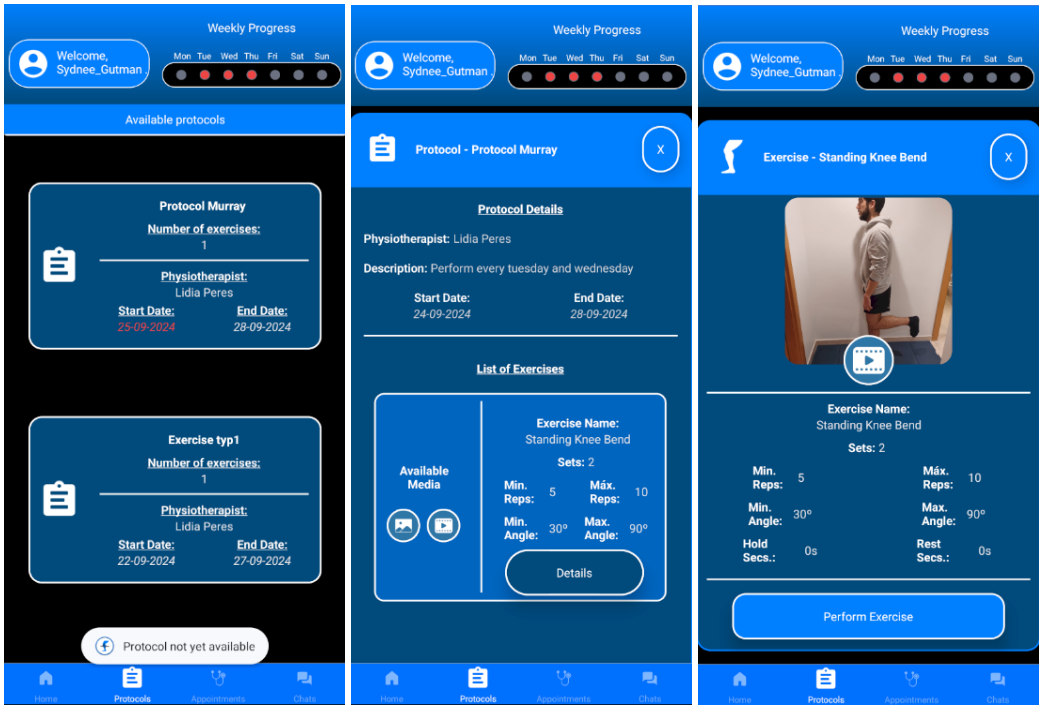
The main focus of the patient application is to provide patients with rehabilitation protocols and their associated physiotherapy exercises.

When a patient accesses the Protocols tab, he has an overview of all the protocols assigned to him, both expired and active ones, and also a brief description and the total number of exercises that is comprised by the protocol as seen in Figure 4.13a. In this screen the protocols by which the start date hasn't been reached yet cannot be opened by patients until the target date is reached. Also, if the end date has already been reached, the patients can no longer perform exercises that belong to that protocol and can only view its details.

By accessing a protocol, the patient can view the full details regarding it, and can also navigate over the list of exercises that are linked to the protocol. In each item from the list of exercises, the patient can check if the exercise has an image or video attached, and can also view some brief details about the exercise, specifically the exercise name, sets, reps and angle thresholds as depicted in Figure 4.13b.

When accessing an exercise in specific, the patient can view in full detail the exercise parameters, preview the image and video associated to the exercise, and also view its exercise readings (if any are available) as seen Figure 4.13c. It's also from this screen that

the patient can initiate the exercise routine, including performing the sensors calibration and also the exercise.



(a) (b) (c)

Figure 4.13: View protocol and exercise screens. (a) Protocol List screen, (b) Protocol details screen, (c) Exercise details screen

A patient by accessing the exercise readings, can then select which exercise session he wishes to view, and can also export that session's data to a .pdf report, comprising all the readings captured in the performed exercise as well as some statistics.

4.5.4 Exercise calibration and performing an Exercise

Before engaging in an physiotherapy exercise, the patient must first perform an exercise setup routine in the application. This setup routine is comprised of three steps: the wearable sensor prototype pairing, the angle mode selection and the calibration routine as seen in Figure 4.14.

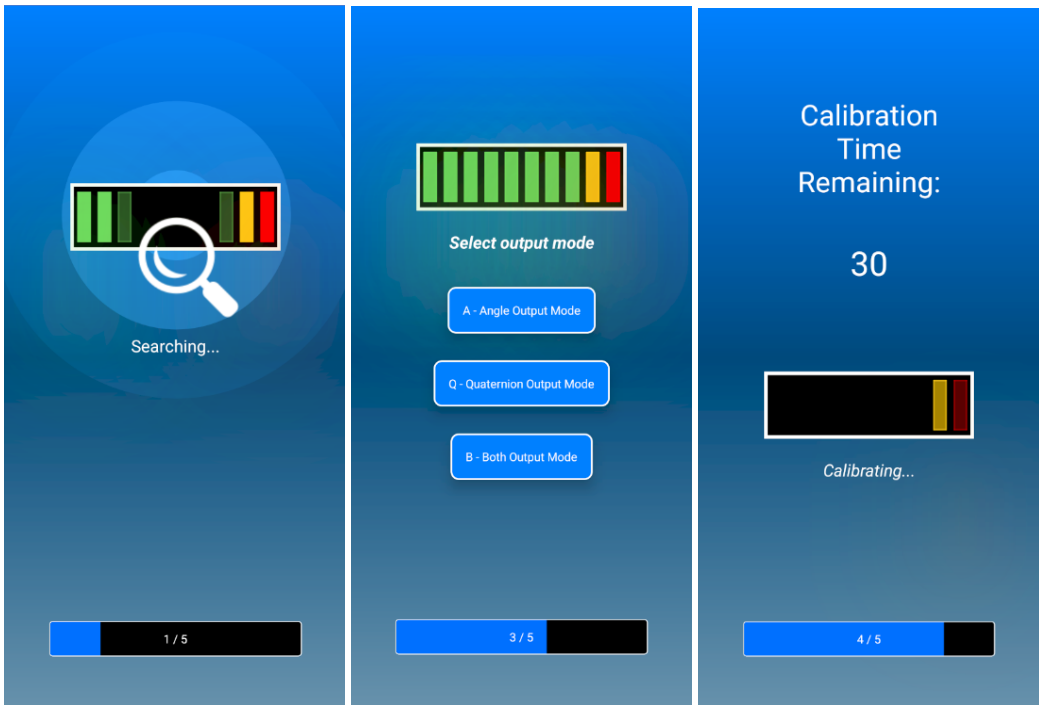
For performing the exercise setup routine as well as performing the exercise, a set of Android permissions were also required: **`BLUETOOTH`**, **`BLUETOOTH_ADMIN`**, **`ACCESS_COARSE_LOCATION`** and **`ACCESS_FINE_LOCATION`** permissions for supporting versions bellow Android 12, and **`BLUETOOTH_SCAN`**, **`BLUETOOTH_CONNECT`**, **`BLUETOOTH_ADVERTISE`**, **`ACCESS_BACKGROUND_LOCATION`** for Android 12 and above.

The routine begins with a prompt asking the patient to scan for the *KNEEMOR* sensors. After finding and connecting to the sensors, the application then prompts the patient to select the desired output mode from the sensors. Afterwards a confirmation of the maximum angle threshold is presented, and after the patient confirming said angle, it

is sent to the wearable sensor prototype via BLE, so it can be used for calculating the current LED bar progress as he performs movement with the knee in the exercise. After this step, the calibration routine begins for determining the starting zero reference of the exercise and also for calculating the offset angle that is required to subtract from the raw readings. This routine ensures that the readings are more accurate and accommodated to different leg anatomies, since different people might have a larger or smaller anatomical leg coverage areas, which can directly impact the sensor readings.

Despite the sensors already offering some internal calibration, this application level calibration is also necessary since it calculates the offset angle value that must be taken from the sensor readings, so that it approximates the angle readings to the real world readings.

To perform this calibration, the patient must first position himself in the starting position of the exercise (with the leg extended) and hold still for thirty seconds. During these thirty seconds, angle readings are being collected from the sensors into an array in the patient application, and when this calibration period ends, an average is performed over the collected values to obtain the offset value that must be taken from each reading.



(a) (b) (c)

Figure 4.14: Exercise setup routine. (a) Searching sensors screen, (b) Angle mode selection screen, (c) Calibration screen

During the calibration and also during the performance of an exercise, it was detected that the sensors sometimes had spike readings which were erroneous, and by considering these erroneous values in the average, it impacted the offset angle that had to be removed in each reading.

In light of this, before performing the average over the non-erroneous readings, it was necessary to first filter out the erroneous values from the collected readings. To achieve this, the Z-score methodology was used, which detects how far away a value is from the mean in terms of standard deviations [73]. By calculating the average and the standard deviation over all the collected values and also setting a Z-score threshold (which determines how far the value can be from the mean), we can then test each collected value by subtracting the calculated mean and then dividing by the standard deviation. If result is greater or equal to the defined Z-score threshold, then the angle reading is considered an outlier and should be discarded from the calibration mean.

After the calibration mean is performed, the exercise then begins. When performing an exercise, the patient obtains a visual feedback from the LED bar mounted on the leg and additionally on the application with a gauge component, as he performs movement with his knee joint during the exercise. When he reaches the maximum threshold defined for the exercise, for a repetition to be accounted for, he must hold the position for five seconds. During the five second window, the same average process used in the calibration routine is performed, hence the resulting average corresponds to the respective repetition angle value. If the average is off by at least four degrees then the repetition is counted as a fault. The application performed angle readings over the readings characteristic every 300 milliseconds.

The reason for this time interval between readings was due to the fact that the BLE library on average could only perform complete a reading request over this period, despite the wearable sensor prototype having a superior output frequency rate, the patient application needed to be throttled accordingly.

Additionally, if the hold seconds parameter would be changed to a different value than 0 seconds, then the patient besides holding the default five seconds window for validating the angle, would have to additionally hold the position for the specified extra seconds.

Upon a repetition being recorded, the patient must then return to the rest position of the exercise. If the rest time is also specified, after reaching the rest position, the patient must also rest for x amount of seconds before performing the next repetition.

When ending the exercise, the various exercise readings are sent to the backend and an exercise completion is created accordingly in the database for future retrieval and reviewing either through the applications UI or through the available exportable reports.

4.6 Conclusions

The *KNEEMOR* system was developed incrementally by phases: first the Sensor Layer was developed, by mounting and assembling the Wearable Sensor Prototype and by ensuring that the angle readings had a appropriate accuracy. Afterwards the backend layer and frontend layers began being implemented in parallel. The backend layer implementation started with database design, afterwards defining the project structure for adhering to the Clean Architecture, and then defining incrementally each REST API endpoint for

supporting each feature. As each endpoint feature was being created, the respective frontend pages in the physiotherapist application were also created along with some tests performed. The Notifications SignalR Hub was implemented in mid-development of the backend when at least the physiotherapist application was mostly complete and some patient screens were already implemented, in order to test the chat functionality. Finally, the patient application was developed last since most of the code could be easily re-adapted from the physiotherapist application due to both applications sharing similar code bases.

The biggest challenges faced during the implementation phase were developing the patient's exercise screen as well as the protocol creation page from the physiotherapist, as well as accommodating the sensor prototype in a small 3D printed box. The patient's exercise screen was difficult to implement due to having to ensure that the various constraints were respected for the exercise and also guaranteeing that the angle validation and rest times were respected with the wearable sensor sending data at all times.

The final system architecture with all the various technologies in place can be viewed in Figure 4.15.

The overall development of the *KNEEMOR* system was executed in a local environment with all its components sharing the same Wi-Fi network. The backend along with the physiotherapist application, were both hosted on the same machine, while the patient application was installed on a mobile phone running Android 12.

After performing various end-to-end tests on each application and validating that sensors readings from wearable sensor prototype were appropriate and also having the three applications in a stable state, the search for tests subjects began and the evaluation process from the next chapter commenced.

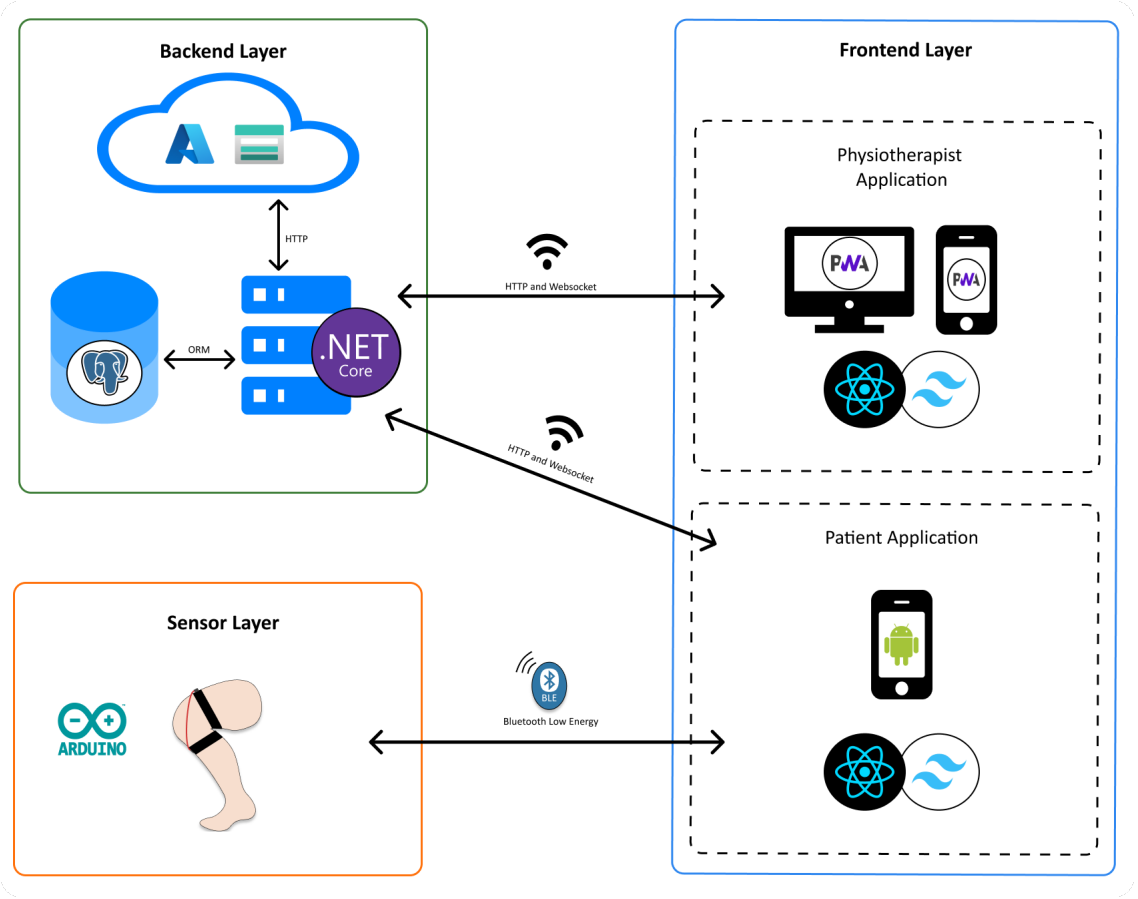


Figure 4.15: Final System Architecture

5

Evaluation and Results

In this chapter the evaluation methodology and the respective results are explored and discussed. It is important to test the system as a whole and ensure that the wearable sensor prototype provides approximate real-world readings and that it is stable during its use. In the first section, a Ruler Test is performed in order to evaluate the sensor's accuracy and effectiveness over four different angles and also to extrapolate the root squared mean error between a real-world angle reference and the sensor readings from the prototype. The test simulates knee flexion and extension using a small goniometer, where one sensor is placed on one ruler representing the thigh segment, and the other sensor is placed on the second ruler to represent the shank segment.

The second section, the readings effectiveness is tested against the application by testing it on people's knees as they flex and extend their knee joints. In this test the mean error from the sensor readings is measured again to ensure there are no significant deviations from the results of the previous test.

The third section concerns Usability Tests that were performed. Here, a set of user stories are defined for both the patient's and physiotherapist's application perspectives. Based on these user stories, some questionnaires are presented to the respective users in order to assess the system's effectiveness and usability.

Finally, the overall cost for the wearable sensor prototype is provided.

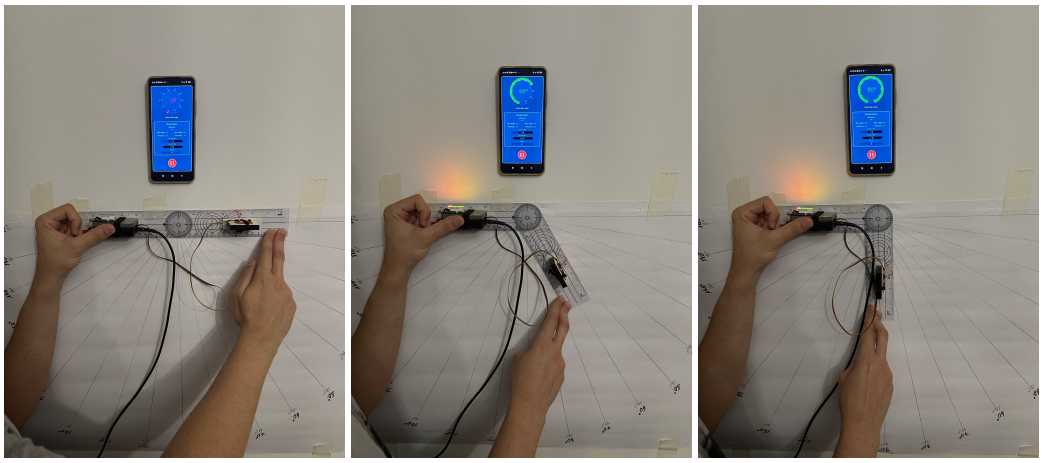
5.1 Ruler Test

The Ruler Test consists on measuring the sensor's accuracy and effectiveness without taking the leg's coverage area into account as a variable for the sensor's calibration and readings.

For this test, the malleable leg sleeves were removed from both sensors to free up space. Once detached from the sleeves, the sensors were then mounted in a small goniometer using double sided tape to secure them to each ruler.

With the sensors attached securely to the goniometer, the goniometer was then placed at the center of an A3-sized cardboard marked with various angle references. The goniometer

was aligned so that its center corresponded to the origin of all angle references, simulating the knee joint. In this setup, the ruler holding the development board and the first IMU represented the thigh segment while the other represented the shank segment. Having this setup allowed for performing the correlation between the angle readings provided from the sensors and the real-world angle references as seen in figure 5.1.



(a) (b) (c)

Figure 5.1: Ruler Test using a Goniometer mounted on a angle reference.
(a) 0 degrees position, (b) 60 degrees position, (c) 90 degrees position

To validate the accuracy of the sensors, four sets of readings were taken into account, each lasting thirty seconds at four distinct angles: 0 degrees, 30 degrees, 60 degrees and 90 degrees. The thirty seconds time frame that was chosen was to provide a longer exposure of the prototype at each angle marking in order to assess if the readings would be stable during this period of time with slim deviations.

To capture the angle readings, a debug flag was turned on the sensor's Arduino code to output to the serial monitor both the raw angle data (i.e. the flexion angle already calculated without considering the calibration from the patient application) and the timestamp (in milliseconds) when each reading started. PuTTY was then used to connect to the prototype and also to log the serial monitor contents to file.

Also as mentioned in chapter 4, in the developed work there are two calibrations, one provided from the sensor's digital motion processor, and another that is performed at the patient's application, for calculating the offset angle regarding the leg's coverage area.

To perform the application calibration, a mobile phone running Android 12 (API 31+) with the *KNEEMOR* patient application was used to select an arbitrary protocol and exercise. This allowed the calibration step to be performed while the ruler was fully extended at 0 degrees and also provided the calibration offset angle to subtract from the raw angle readings.

Once the calibration was completed and the exercise began, the goniometer's shank ruler was moved to each target angle and held in place for thirty seconds in order to collect the sensor's readings to log.

Each thirty second angle collection window yielded approximately 600 angle readings from the prototype which were then used for analysis and comparison.

Additionally, since the wearable sensor prototype only produced the raw angle values, a distinction is made in the results between the raw angles and the calibrated angles, where the latter takes into account the calibration angle offset. The comparison can be seen in figure 5.2, where each set of readings has a pair, one pertaining to the raw angle readings and another for the calibrated readings.

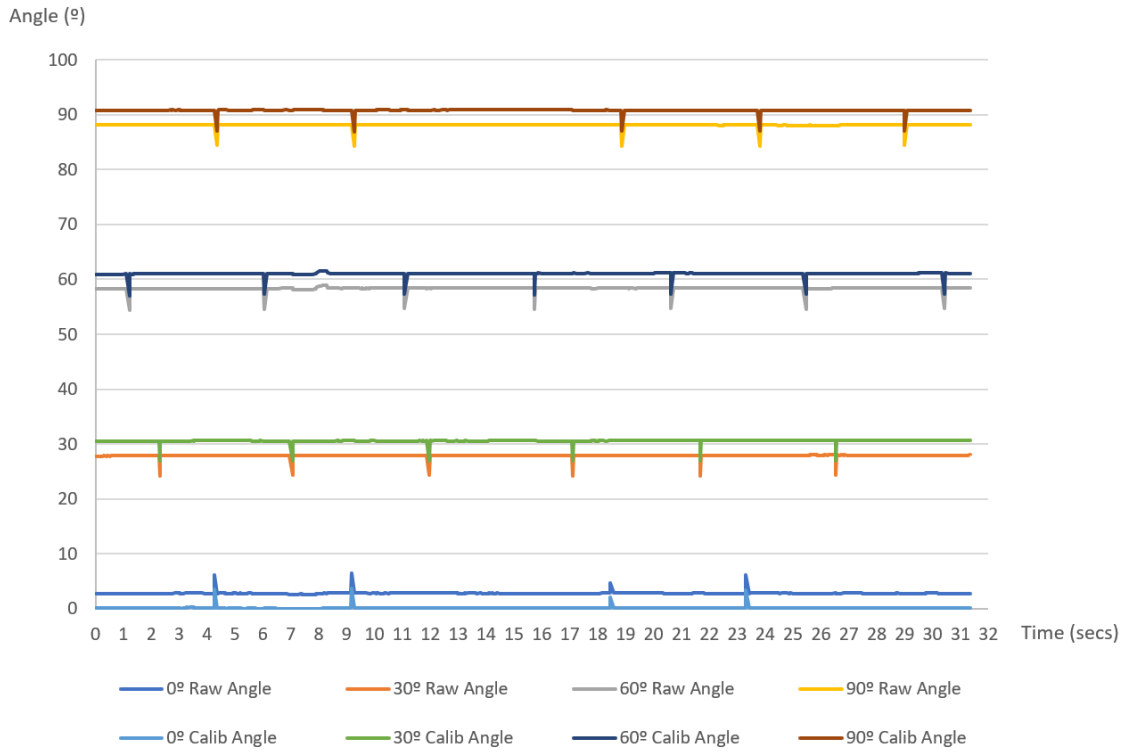


Figure 5.2: Ruler Test Results - Comparison between raw and calibrated angle readings

From the analysis of the graph, it's clear that the calibrated readings converge to targeted angles with only a few readings having some minor fluctuations. The reason for these fluctuations is unclear, but it could be related to magnetic disturbances on the magnetometer, or due to some manufacturing defects from the sensors. These fluctuations are mitigated by using the Z-score method as discussed in chapter 4, when performing repetitions in exercises, but since in this test the only concern is with the angle readings provided from the sensors and not undergoing a particular exercise, these outlier values are more prevalent.

To support the analysis of the graph, the root mean square error (RMSE) was also calculated for each set of angle readings and summarized in Table 5.1, depicting both raw and calibrated RMSE values for each target angle. The root mean squared error tells us how far the captured angle readings are from the expected real-world angles. The lower the RMSE is, the closer the angle readings are to the reference angles. From the

Angle	RMSE Raw Angles	RMSE Calib Angles
0º	2,82	0.29
30º	2,15	0.67
60º	1,70	1.10
90º	1,90	0.88

Table 5.1: RMSE Calculations for targeted angles

table we confirm that the calibration angles are in fact closer to the real-world angles in comparison to the raw angles, it's concluded that the application level calibration has a positive outcome on providing angles that are closer to the reality.

5.2 Knee angle measurement evaluation when performing exercises

In this test, the knee angle measurements were performed following an exercise routine from the patient application. The readings collected in this test were angle readings averaged from repetitions accounted in a exercise. The three proposed exercises mentioned in chapter 2 were used to perform this test and 7 repetitions were collected from each exercise with three distinct angles for each of them.

Additionally, during the performance of this test, with the aid of a physiotherapist and a goniometer, the measured angles angles by the application were also compared against the goniometer measurement made by the physiotherapist.

The three elected angles for each exercise were, 90 degrees for heel slides, 60 degrees for standing knee bend and 84 degrees for knee extension.

In the heel slide exercise, the application read the correct angle accordingly and also the physiotherapist through the goniometer also had the same reference as seen in figure 5.3a. During the performance of the exercise itself, in the 4th repetition, there was slight spike which was due to the heel having slided too fast and caused several spike readings in the process, which then impacted the average for that repetition and caused this repetition to be marked as a fault in the application. The rest of the exercise repetitions were within the accepted angle threshold as the repetition readings didn't deviate further away from 3 degrees of the targeted angle as seen in the blue markings in figure 5.4.

The results regarding the standing knee bend exercise, the initial readings were a bit deviated from the expected angle, the reason for this was that it was hard to maintain the shank in the correct angle over the duration of the 5 seconds to perform the repetition validation, which caused the first three readings to be off by more than 4 degrees, specially the second repetition which was recorded with 71.07 degrees as seen figure 5.4. The remaining readings were just off by 2 to 4 degrees from the expected angle and these were considered not a fault in the exercise as they were within the acceptable threshold.

The last exercise regarding the sitting knee extension, the resulting repetition readings converged to the expected angle with slim deviations, with the exception of the first

(a) (b) (c)

Figure 5.3: Knee angle measurement during exercise performance.
(a) Heel slides measured by a physiotherapist and a goniometer, (b) Standing knee bend, (c) Sitting knee extension

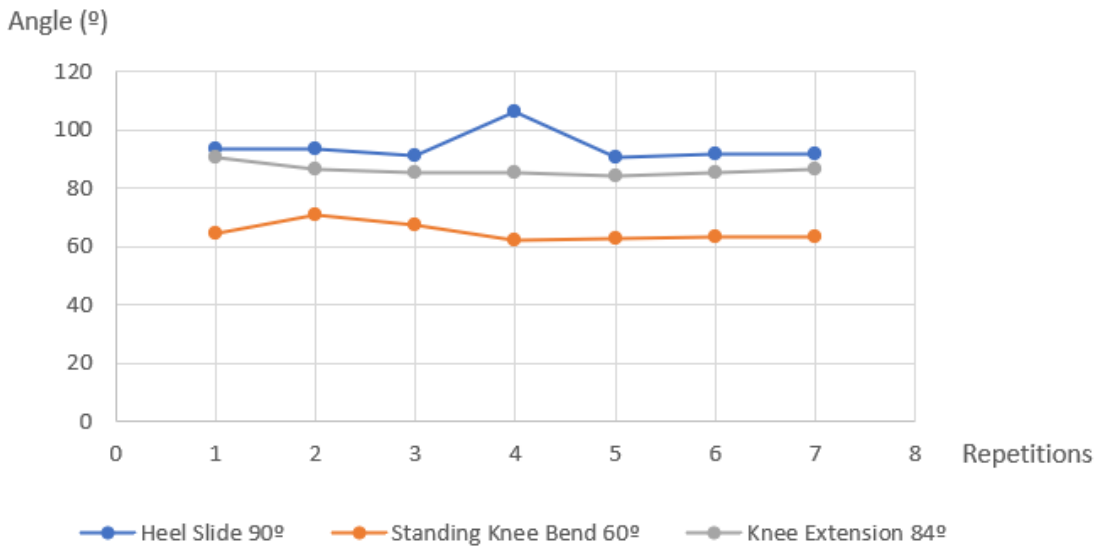


Figure 5.4: Knee angle measurement over developed work exercises

reading that was off by 6 degrees. It was through the calibration of this exercise that it was discovered that for all exercises it was required to calibrate the sensors with the leg fully extended, otherwise the readings would be incorrect. In this exercise in particular, to have the calibration working with the knee flexed as the starting calibration position, it would require some additional logic to tell the patient application to treat this exercise differently in terms of the angle it has to subtract from the readings. Since in the current application and prototype conditions, starting with the knee flexed during the calibration routine, would result in a 0 degree reading with the knee flexed. However, when extending the knee to the target angle, the readings would not be correct and the readings would also be negative.

From RMSE calculations performed on the repetitions of the three exercises, the error between the real angle values has slightly increased as seen in the Table 5.2, but this is expected as these readings now have the anatomical leg area into account, and also since holding the angle positions in the target angles is also harder to maintain compared to the goniometer where the sensors were well fixated and stabilized.

Exercise and Angle	RMSE
Heel Slide 90º	6.58
Standing Knee Bend 60º	5.79
Knee Extension 84º	3.01

Table 5.2: RMSE Calculations for targeted exercises

Despite the negative results on some of the repetitions, the majority of them were acceptable and within the desired angle thresholds, with the only problem being detected were the spike readings that randomly occurred throughout an exercise.

5.3 Usability Tests

When evaluating the *KNEEMOR* system, a set of usability tests were performed in order to assess the system's overall usability from the perspective of the patient and also from the perspective of the physiotherapist.

The patient application should run on Android version 12+ (API 31) due to the direct dependency on the BLE library used on React Native and due to most of the development and the test having been performed in that version. The physiotherapist application should run on Chrome web browser, preferably on a desktop so that the physiotherapist has the best user experience, however it is also available for mobile browsers but some screens might not be fully responsive resulting in a worse user experience.

These usability tests rely on user stories portraying a chain of tasks or events that each user should follow. By analyzing the user's behavior and getting their respective feedback when they are undergoing these tasks, we can obtain valuable insights on how the system should behave or if the system is well tailored to address the patient's and physiotherapist's needs.

Supporting these tasks and to also gather the respective feedback from each user type, a set of questions were administered following the system usability scale (SUS) methodology [16]. This methodology provides a simple and abstract way of measuring a system's usability by rating 10 mixed questions (positive and negative questions) from a Likert scale of 0 to 5 (strongly disagree to strongly agree). The answers are then converted into a usability score ranging from 0 to 100 which indicates the overall usability of the underlying system.

5.3.1 Patient

The patient's user stories are focused on tasks that involve the patient interacting with the mobile application and also undergoing physiotherapy exercises while wearing the wearable sensor prototype. The selected user stories comprise the main functionalities of the patient application and provides a baseline for the questionnaire that was proposed.

5.3.1.1 User stories and questionnaire

The user stories as previously mentioned, depict a set of tasks that the patient must follow through until reaching the end of the user story. For conducting these user stories, a **Android phone running Android 12** with a pre-installed version of the KneemorApp was provided for the test subjects and also the wearable sensor prototype. The elected user stories for assessing the patient application were the following:

1. **Logging in and booking an appointment.** The patient wants to open the KneemorApp and book an appointment. Opening the app he must log in by using the provided credentials user "patienttest" and password "Kneemortrial1#". Upon logging in he should be able to see his weekly progress, upcoming exercises and upcoming appointments. The patient then navigates to the menu "Appointments", and selects the option "Book Appointment". A view should then be displayed containing a dropdown component which comprises the list of available physiotherapists of the system, as well as a calendar portraying the availabilities. The patient then selects a day that is available for appointment and confirms the location and the time slot. The user story ends when the appointment booking is finished and the patient is then redirected to the respective appointment details view.

2. **Applying the wearable sensor prototype and conducting an exercise from a protocol.** The patient wants to open the KneemorApp and conduct a physiotherapy exercise from a specific protocol. Upon opening the App, if he hasn't logged in, he must log in using the credentials user "patienttest" and password "Kneemortrial1#". By concluding the login process, he should then navigate to the "Protocols" menu, then access the option "View Protocols". The patient then proceeds to select the protocol named "Test2" which holds 2 exercises. After selecting the protocol, he then scrolls down to the exercises section and selects the standing knee flexion exercise so he can start conducting the physiotherapy exercise. He reviews both the exercise image as well as the exercise video to get an idea of how to perform the exercise. After reviewing the exercise media, he then presses the start exercise option and follows along the calibration phase. The user story concludes when the patient starts performing the selected exercise and finishes the underlying sets.

Based on these two user stories, a set of twelve questions were asked to test subjects regarding the application's usability in terms some of the functionalities, more specifically,

appointment booking, protocol and exercise viewing, the comfort and ease of use of the wearable sensor prototype during an exercise and also possible future improvements that could be added to the application (to view questions in detail, see annex I.1).

5.3.1.2 Results

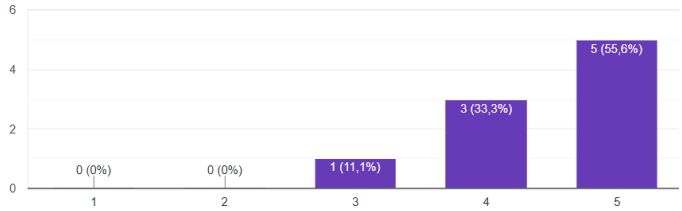
The patient tests were performed on healthy knees of nine test subjects, with ages ranging from 27 and 66 years old.

The usability tests regarding the patient application, were used to assess if the application was intuitive for patients to book and view appointments, conduct their protocol exercises and also to receive feedback from the built prototype that measures the knee angle when performing exercises.

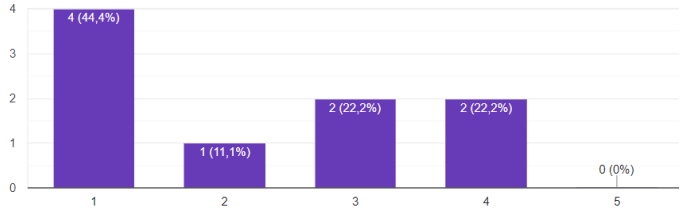
The first four questions were concerned about the appointment viewing and booking in the patient's application. The navigation around the application and the overall booking of appointments was considered an easy task for all test subjects as seen in figure 5.5, however, in some application screens, specifically viewing an already booked appointment, the test subjects struggled to find a way to return to the previous screen on the appointments navigator. This happened since there wasn't an explicit button to return to the previous screen, and instead, the test subjects needed to do an additional tap on the Appointments tab button, in order to return to the navigators root screen. Three test subjects found that calendar component was a bit confusing due to not instantly rendering the available days, and due to requiring the patients must tap on a small arrow icon to expand the calendar initially view the appointment day slots. One of the test subjects also suggested that the application should have the iconography as well as the typography standardized, since some screens weren't uniformed with the rest of the application. These small UI changes were then later on included in the final developed work.

The remaining questions were concerned about protocol and exercise viewing, as well as using the exercise routine in conjunction with the wearable sensor prototype, and additionally, potential improvements for the patient application. These questions evaluated the usability of the underlying exercise screens, the pairing process, the comfort provided from the wearable sensor prototype, and lastly, the usefulness of the feedback system provided by both the patient application and the wearable sensor prototype. Overall, the protocol and exercise screens were easy to follow along for the patient subjects as seen in graph results from the figure 5.6, however, some issues were detected with the wearable sensor prototype in some test subjects. For two test subjects, the detachable thigh sleeve had to be very tight due to these two test subjects having a larger leg diameter, which caused some discomfort with the wearable sensor components being very compressed into the thigh area. The suggestion made by both subjects, was to use a thicker material so that the thigh wouldn't feel the components as much and to extend the sleeve a bit more to accommodate larger leg diameters.

For other three test subjects, whom had thighs with a conic shape (meaning they had



(a)



(b)

Figure 5.5: Graph results regarding the questions about the ease of navigation on appointment screens and the usability of the calendar component during the appointment booking process.
(a) Screens navigation results, (b) Calendar component usability results

a larger diameter in the upper thigh region and a gradually narrower diameter closer to the knee's patella), the detachable thigh sleeve started falling down the thigh during the standing knee bend exercise. Their suggestion was to add a small Velcro strap to adjust near that area to prevent the sleeve from slipping off the knee.

Additionally, two test subjects found that having the LED bar provide feedback to the user during an exercise was unnecessary, as the user could view the information through the patient's application. The LED indicators were considered only useful during the pairing process and also turning on the sensors, but not for exercise performance.

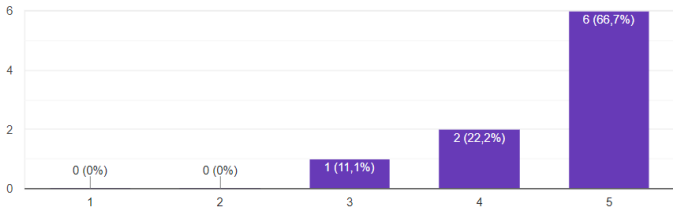
The average SUS score from all test patient test subject responses was 76.38%, indicating that the patient application has good usability metric.

5.3.2 Physiotherapist

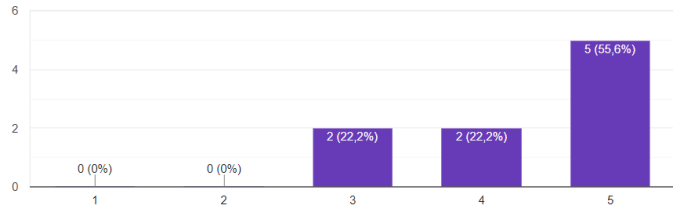
The physiotherapist's user stories are instead focused on tasks regarding exercise type creation and protocol assignment to patients and also appointment management. The selected user stories comprise the exercise type creation, protocol assignment to a patient and confirmation of an appointment booking.

5.3.2.1 User stories and questionnaire

For conducting the following user stories, a regular laptop with Google Chrome was provided so that the test subjects acting as physiotherapists could have a desktop experience from the KneemorPWA. The elected user stories for assessing the physiotherapist application were the following:



(a)



(b)

Figure 5.6: Graph results related to questions regarding the intuitiveness of the exercise setup and calibration phase screens, as well as the comfort of the wearable sensor prototype.
(a) Exercise setup results, (b) Wearable sensor prototype comfort results

1. **Creating an exercise type.** The physiotherapist wants to open the KneemorPWA and create a new exercise type. Opening the PWA on the browser, he must log in by using the provided credentials, user "physiotest" and password "Kneemortrial1#". Upon logging in he should be able to see his upcoming appointments as well as the monthly appointment attendances. The physiotherapist then navigates to the "Protocols" menu, and selects the option "Create Exercise Type". The physiotherapist fills out the exercise type metadata, checks "Provide multimedia data" and presses "Create Exercise Type". Afterwards a file input is displayed and the physiotherapist must select a image and a video from his hard drive. Upon confirming the selection of the image and video from the file input, a preview should be displayed for both media types. The user story is concluded when the physiotherapist clicks on the option "Add multimedia content" and returns to the "Protocols" view.

2. **Create a protocol and assign it to a patient.** The physiotherapist wants to open the KneemorPWA and create a new protocol and assign it to a patient. Opening the PWA on the browser, if he isn't logged in yet, he must log in by using the provided credentials, user "physiotest" and password "Kneemortrial1#". Afterwards the physiotherapist must navigate to the "Protocols" menu and click on the button "Create new Protocol". Upon entering the create new protocol view, he must now fill out the protocol metadata and select the patient "patienttest". Scrolling down to the exercises section, the physiotherapist will fill out the exercise parameters and select the exercise schedule to conduct the parameterized exercise. This user story concludes when the physiotherapist clicks on the button "Create Protocol" and is redirected to the "Protocols" view where he can now see the created protocol in the

protocol table.

3. **Confirm pending appointment from patient.** The physiotherapist wants to open the KneemorPWA and confirm a pending appointment from a patient. Opening the PWA on the browser, if he isn't logged in yet, he must log in by using the provided credentials, user "physiotest" and password "Kneemortrial1#". The physiotherapist then clicks on the pending appointment from his upcoming appointments list. Upon opening the appointment view, he then confirms the appointment by clicking on the button "Approve". The user story concludes when the physiotherapist navigates to the "Chats" menu and can now see the newly created chat regarding the patient that created the appointment.

4. **Attach appointment data to an appointment and export report to .pdf format.** The physiotherapist wants to open the KneemorPWA and attach appointment data to a patient's appointment. Opening the PWA on the browser, if he isn't logged in yet, he must log in by using the provided credentials, user "physiotest" and password "Kneemortrial1#". The physiotherapist navigates to the Appointments menu and selects the appointment named "test1" from the appointments table. After entering the specified appointment, the physiotherapist scrolls down to the AppointmentData section and clicks on the "+" button. The physiotherapist fills the form data and submits the form. The user story concludes when the physiotherapist scroll down the AppointmentData page to the export as section and clicks on the PDF icon to download the appointment data.

Based on these four user stories, a set of twelve questions was asked to test subjects, in order to gather insights and feedback from the physiotherapist application (to view questions in detail, see annex I.2).

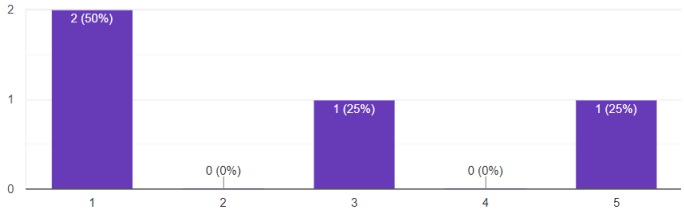
5.3.2.2 Results

The physiotherapist's tests were performed by a total of four professionals working in the physical therapy practice. The provided usability tests were used to assess if the physiotherapist application had the needed functionalities for supporting the daily operational activities in the physiotherapy practice, while also assessing if the application itself was intuitive in the perspective of the professionals. Additionally, these tests determined whether the protocol creation, exercise creation, and assignment flows were considered a complex tasks, and if the reports generated by the application were useful in supporting the physiotherapy practice and monitoring the patient's overall progress.

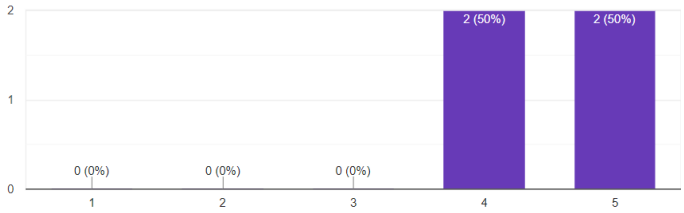
Regarding the first four questions related to exercise and protocol creation and assignment, it was identified that the segregation between ExerciseTypes, Exercises and Protocols was a bit confusing, due to the fact that to create a Protocol, there is always a need for at least one ExerciseType to exist in the system. The name of the entity could have been changed to ExerciseMetadata or ExerciseMedia to avoid confusion, since its main

purpose is to assign only the exercise name, description and some multimedia content to explain the exercise. Due to this, two physiotherapists evaluated both ExerciseType creation and Protocol creation negatively, as the user interface was very intuitive in this regard. However, the parameterization of exercises during protocol creation was evaluated positively, as seen in figure 5.7, since it provided versatility in creating different types of exercises and setting different angle thresholds for patients to achieve.

The limitation of the exercise schedule to a single time period was not a major concern for physiotheparists. One professional even stated that the most important factor is ensuring that the patient performs the exercise during the day, without the need for a strict exercise schedule.



(a)



(b)

Figure 5.7: Graph results related to the questions on protocol creation and exercise parameterization.
(a) Protocol creation results, (b) Exercise parameterization results

The following four questions addressed the process of confirming a pending appointment, triggering a new chat between a patient, and whether notifications were considered an asset or an annoyance. The results from these questions were mostly positive, with the notifications being regarded as an asset rather than annoyance or cluttering feature on the application. Regarding the chat creation process, one physiotherapist suggested it would be useful to allow interaction with fellow colleagues from the same clinic, and also proposed that chat creation could be manually initiated by the physiotherapist's end rather than automatically. Since there is no way to block or remove a chat, it could become an annoyance if patients abuse the chat system.

The last set of questions focused on the reports generated by the application and potential improvements. All physiotherapists found this functionality very useful for both the physiotherapists and patients. One physiotherapist suggested automatic summarization of the results provided from the exercise readings report, as well as the protocol monthly report. This would enhance boost productivity and remove the need for manually analyzing and scrutinizing the resulting data.

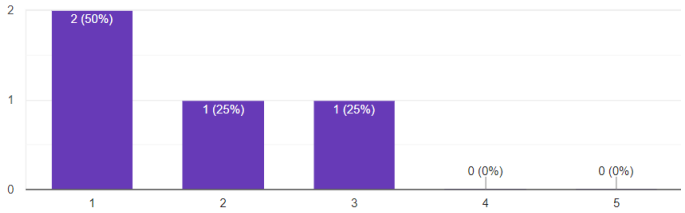


Figure 5.8: Graph with the results concerning the reports usefulness for tracking the knee range of motion progress

In addition to the improvements already mentioned, two new parameters for exercise parameterization were suggested by two physiotherapists. One suggested the addition of a weight load parameter, as the patient could demonstrate knee joint mobility without load, but with load, the muscle strength might be impacted, thereby affecting knee mobility. The other suggestion was to include the expected real zero reference of a patient's fully extended leg. This parameter wouldn't be used directly in physiotherapy exercises but would differentiate the sensor's zero reference from an injured knee, from the expected zero reference of a healthy knee.

Additionally, another physiotherapist also proposed introducing a pain scale for the patient to complete after finishing an exercise. This metric would help physiotherapists determine whether the current exercise parameters need to be adjusted or if a different exercise should be assigned. Lastly, it was also mentioned that the sensors should be ready for exercises beyond those developed in the thesis work, such as lunges and squats.

The average SUS score from all physiotherapist responses was 71.87%, which indicates that the physiotherapist application has good usability but barely meets the threshold, as the lower threshold for considering bad usability is 68%.

5.4 Wearable Sensor Prototype Cost Analysis

As part of the sensor layer requirements, the wearable sensor prototype should be cost effective, meaning the underlying components should be cheap and easy to acquire, so there is a wider adoption and accessibility to this prototype in the physiotherapy practice.

The prototype cost estimations are depicted in Table 5.3 and prices refer to the year 2023-2024, which was when most components were acquired. These prices as of today might have changed and might be cheaper, as the shortage of manufacturing of microcontroller chips and other components begins to stabilize with the conclusion of the COVID-19 pandemic. The biggest cost regarding the prototype were the two Sparkfun ICM-20948 IMUs, due to being relatively new in the market and also for providing integrated sensor fusion algorithms in their chips. Other IMUs could've been considered, reducing the overall prototype cost but the aim of the thesis was also to explore these new IMUs and their capabilities compared to more researched and experimented IMUs.

The power bank was fairly expensive but it was a personal choice due to its small size,

brand and battery longevity. The remaining components weren't as expensive as the ones mentioned, and could be acquired easily.

Quantity	Component	Cost
2x	SparkFun 9DoF IMU Breakout - ICM-20948 (Qwiic)	20€
1x	ESP32 DevKit-C WROOM	12.50€
1x	Seeed Grove LED Bar	6€
1x	Qwiic Cable - 500mm	3.69€
1x	Qwiic Cable - 4 pin breadboard jumper	2.09€
1x	Anker PowerCore 5000 Power bank	20.32 €
1x	deleyCON 0.15m Micro USB Nylon Cable	4.86 €
1x	Detachable waistband	6 €
1x	Velcro parts	1.30€
	Total	**90.76€**

Table 5.3: Wearable Sensor Prototype - Cost Estimation

5.5 Conclusions

For assessing the system's usability and accuracy, it was necessary to perform various tests accordingly, including evaluating the wearable sensor's prototype accuracy, and gathering insights and feedback from test subjects.

During the evaluation of the wearable sensor prototype, it was established that the application level calibration was crucial for obtaining angle readings that were closer to reality. Additionally, during test trials, it was detected that to achieve accurate readings, the initial start position should always be with the leg extended, as the developed system is not yet prepared for other types of initial positions for calibration.

The placement of the sensors should be always be on the anterior sides of both the thigh and shin. This decision was made to provide the patient with visibility over the LED bar and also because the developed algorithm for calculating the knee joint angle relied on these specific IMU placement references.

The usability tests were generally positive for both applications, and several suggestions for improvements were made and included in future work, while others were implemented towards the end of the developed work. While the patient application provides some autonomy for performing physiotherapy exercises, it does not replace the need for professional physiotherapy practice, as evaluation, diagnosis, and treatments should always be provided by a trained professional. The application is instead to serve as a tool to aid in the physiotherapy practice.

The prototype itself met its intended goals, as it is convenient, low-cost, and provides acceptable measurement accuracy for the knee angle. Because the prototype has a low production cost, it can be easily accessible for physical therapy practitioners and clinics looking to introduce remote and autonomous physiotherapy into their practice.

6

Conclusions

This chapter covers the conclusions and insights that were gathered throughout the development, evaluation and external feedback from test subjects using *KNEEMOR*. Based on these conclusions, in this chapter it is also explored a list of future improvements and recommendations that have been taken into account, for instilling a more robust, accurate and usable system for the physiotherapy practice.

6.1 Conclusions

With the rise of AI adoption and also the rapid evolution of technology around the world, new technologies and solutions have emerged to address real-world problems. Additionally, with the continuous reduction in the costs of electronic components, sensors, and microprocessors, more IoT technology is being developed and explored, particularly in medical services and practices that involve physical interaction.

The goal with this dissertation was to develop a system that does not replace the practice of physiotherapy but rather supports it, allowing patients to have some degree of autonomy and receive instant feedback when performing their exercises without direct physiotherapist supervision. For physiotherapists, it provides a remote, cost effective option for delivering physiotherapy services while also monitoring capabilities over their patients.

The *KNEEMOR* system was built with these goals in mind, specifically tailored to typical operations of physiotherapy clinics, including patient management, appointment scheduling and patient record keeping. The system enables patients to perform physiotherapy exercises from the comfort of their home, book appointments, chat with their physiotherapists, and view or export meaningful metrics related to their knee range of motion progress. Physiotherapists, in turn, can use the *KNEEMOR* system to manage appointments, prescribe treatments, recommendations, and diagnoses, all of which that can be accessed and exported by the patient and by themselves. Additionally, they can manage patient protocols and exercises, adjusting them as needed based on the patient's knee health progress, while also managing their appointments schedules.

The wearable sensor prototype, which was the main emphasis of this thesis, had some issues, particularly with occasional spike readings during exercises, reports from some test subjects that the thigh sleeve would slip sometimes, and the current limitation that exercise calibration requires the leg to be fully extended.

The patient application supporting the exercise performance also had some issues, more specifically on the rate of collecting angle readings from the BLE characteristic. This issue was due to the BLE library that was used in the system, that could only fulfill each read characteristic request in a average of 300 ms, before being able to fulfill a new reading request again, which caused the visual gauge component to have a stutter and delayed effect for displaying the angle readings. Another issue was that the notification service in the patient application, sometimes stopped working. This problem occurred between logging off the application and between bringing the application from the background and foreground. However, the protocol, exercise viewing and also exercise setup screens were properly implemented and provided intuitive navigation as well as nice features to guide patients on their exercises.

The physiotherapist application had a similar issue related to notifications. Sometimes notifications didn't get triggered and required refreshing the application to make notifications work again. Additionally, more visual graph components could've been implemented to provide more information about appointments and patient progress. Protocol creation also could've been more simplified regarding ExerciseTypes, but the overall parameterization of exercises was considered useful and versatile for creating different combinations of exercises for patients. Supporting exercise readings and protocols, the provided generated reports were also considered good features for the application but could be improved with more metrics and provide different types of graphs for displaying the resulting data.

The tests performed during the evaluation of the prototype were considered acceptable, as the measured angles mostly converged with the targeted angles, and the collected readings remained within the defined angle thresholds.

While the system is considered functional in most of its core features, and apart from the limitations mentioned previously, there were also some functionalities that were not fully implemented or had some specific bugs that could not be resolved in time for the final delivery of the work. Despite this, the main objective of providing a solution for measuring the knee joint angle using the new ICM-20948 IMUs was achieved, along with the development of two usable interfaces for both primary stakeholders of the system.

6.2 Future Work

Throughout the proposal, development and evaluation phases, and as the thesis work progressed, new features and improvements arose as time went by. In this section, additional work is discussed and theorized for providing a foundation for improving the KNEEMOR system and expanding it with new functionalities.

6.2.1 Wearable Sensor Prototype

During development of the wearable sensor, as previously was stated, the buzzer was causing some interference in the readings from the IMU's and due to time constraints it was scrapped from the final product. It would be interesting to fix this issue in the future to not only provide visual feedback but also audible feedback from the wearable sensor. Other Arduino libraries could also have been experimented and calibration of the sensors could be improved for providing better accuracy, and less noise in terms of angle readings.

6.2.2 Patient Application

As seen in previous works of other colleagues [62, 19], it would be interesting to integrate in the patient's application a 3D view of a leg which could be animated through the respective quaternion data emitted by the two sensors. This could make the application more engaging and interesting for the patient.

6.2.3 Physiotherapist Application

Due to the scope of the thesis having been wide, and also due to the extra effort of maintaining two distinct frontends, some shortcuts had to be taken in order to accelerate development of the whole system. One of these shortcuts was **discarding the responsiveness for mobile in some pages of the physiotherapist application**. Overall, the various pages were built with responsiveness in mind, but in some components adjustments are required for providing the best user experience for physiotherapists.

In the protocol editing, it was decided for simplifying the development process that exercise schedules should all share the same start and end times for all days of the week. It should be possible for physiotherapists to update only a specific exercise schedule for a given day of the week, making the schedule updates more customizable.

The **caching strategies and offline support** used in the physiotherapist application could also be improved and optimized, particularly one issue that arose during development was the caching of video content, that wasn't being served properly from the cache. Also not all pages are properly cached and continue internet dependent, as future work we could improve these pages and fully support them offline.

From the input gathered from physiotherapists that contributed with testing of the applications, it was also mentioned that it would be interesting to have a functionality to follow along an exercise session remotely in real-time. This functionality is somewhat similar to the exercise playback feature from the colleague's work [19] but with the aspect of displaying the information in real-time instead of replaying the exercise readings. A suggestion to add a weight parameter would be interesting to have, in order correlate the evolution of the knee joint in terms of muscle strength. This was added at the end of the developed work.

6.2.4 Backend

During development, it was decided that all day of week exercise schedules shared the same start and end times. It would be interesting to allow more fine grained control over exercise schedule updates.

Chats and ChatMessages could support **media uploads of pictures and files**. Since we are already using Azure Blob Storage for storing exercise media, profile and clinic images, we could also extend the Chat functionality for also allowing file uploads between patients and physiotherapists.

More informative reports could be generated, and an improvement to the currently available reports would be to allow the clinic image to be inserted in the report and also define a different color theme depending on the clinic's preferences.

Lastly some backend caching could also be introduced, since in the frontend layers there is some caching strategies implemented, in the backend it would be a solid decision to introduce caching into queries that are regularly used.

6.2.5 Deployment

Due to time constraints, the system was mainly tested on a local environment. Since the backend application can already be containerized, as future work we could explore cloud solutions for deploying the containerized backend along with physiotherapist application as well.

Bibliography

[1] *.NET - What is .NET?* URL: https://dotnet.microsoft.com/en-us/learn/dotnet/what-is-dotnet (cit. on p. 38).

[2] *A Complete Guide to Potentiometers*. URL: https://uk.rs-online.com/web/content/discovery/ideas-and-advice/potentiometers-guide (visited on 2023-01-17) (cit. on pp. 18, 49, 50, 53).

[3] K. Z. Adil Ali Saleem et al. "IoT Based Smart Physiotherapy System: A Review". In: *Journal of Engineering Research and Sciences* 1.10 (2022), pp. 45–55. DOI: 10.55708/js0110007 (cit. on pp. 2, 4).

[4] N. T. et. al. "Comparative Study of IoT Development Boards in 2021: Choosing right Hardware for IoT Projects". In: *2021 2nd International Conference on Intelligent Engineering and Management (ICIEM)*. 2021 (cit. on p. 48).

[5] *AMFITRACK - Our technology*. URL: https://www.amfitrack.com/our-technology (visited on 2024-01-02) (cit. on p. 19).

[6] *An Introduction to Positional Tracking and Degrees of Freedom (DOF)*. URL: https://www.roadtovr.com/introduction-positional-tracking-degrees-freedom-dof/ (visited on 2023-02-01) (cit. on p. 70).

[7] *An IoT-Based Smart System with an MQTT Broker for Individual Patient Vital Sign Monitoring in Potential Emergency or Prehospital Applications*. URL: https://www.hindawi.com/journals/emi/2022/7245650/ (visited on 2023-02-06) (cit. on p. 31).

[8] *Anatomical Plane*. URL: https://en.wikipedia.org/wiki/Anatomical_plane (visited on 2022-09-27) (cit. on pp. 8, 9).

[9] R. Antunes et al. "Accuracy of Measuring Knee Flexion after TKA through Wearable IMU Sensors". In: *Journal of Functional Morphology and Kinesiology* 6.3 (2021). ISSN: 2411-5142. DOI: 10.3390/jfmk6030060. URL: https://www.mdpi.com/2411-5142/6/3/60 (cit. on pp. 26, 27).

[10] *ASP.NET Core - Request processing pipeline*. URL: https://learn.microsoft.com/en-us/aspnet/core/fundamentals/middleware/?view=aspnetcore-7.0 (cit. on p. 39).

[11] *Attitude and heading reference system(AHRS)*. URL: https://www.vectornav.com/resources/inertial-navigation-primer/theory-of-operation/theory-ahrs (visited on 2024-01-08) (cit. on p. 20).

[12] *Authentication and authorization in ASP.NET Core SignalR*. URL: https://learn.microsoft.com/en-us/aspnet/core/signalr/authn-and-authz?view=aspnetcore-7.0 (cit. on p. 48).

[13] *Benefits of Remote Physical Therapy*. URL: https://www.fickewirth.com/the-loop-detail.php?Benefits-of-Remote-Physical-Therapy-212 (visited on 2023-02-02) (cit. on p. 3).

[14] T. Bi Irie guy-cedric. "A Comparative Study on AES 128 BIT AND AES 256 BIT". In: *INTERNATIONAL JOURNAL OF COMPUTER SCIENCES AND ENGINEERING* volume 6 (2018-09), pp. 30–33. DOI: 10.26438/ijsrcse/v6i4.3033 (cit. on p. 94).

[15] *BPMpathway - How it works?* URL: https://www.bpmpathway.com/how-it-works/ (visited on 2023-05-14) (cit. on p. 58).

[16] J. Brooke. "SUS: A quick and dirty usability scale". In: *Usability Eval. Ind.* 189 (1995-11) (cit. on p. 118).

[17] *Code in Clean vs (Traditional Layered) Architecture .NET*. URL: https://medium.com/codenx/code-in-clean-vs-traditional-layered-architecture-net-31c4cad8f815 (cit. on p. 40).

[18] *Common Language Runtime (CLR) overview*. URL: https://learn.microsoft.com/en-us/dotnet/standard/clr (cit. on p. 38).

[19] L. Correia. *Development of a web and mobile based system for remote physiotherapy using motion sensors*. 2021 (cit. on p. 129).

[20] *Cyber-Physical System based E-Health: Knee Joint Physical Therapy Monitoring*. URL: https://dergipark.org.tr/en/download/article-file/2084422 (visited on 2023-02-06) (cit. on p. 31).

[21] F. J. Dian, A. Yousefi, and S. Lim. "A practical study on Bluetooth Low Energy (BLE) throughput". In: *2018 IEEE 9th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*. 2018, pp. 768–771. DOI: 10.1109/IEMCON.2018.8614763 (cit. on pp. 54, 55).

[22] D. Eleskovic. "A comparison of Hybrid and Progressive Web Applications for the Android platform". In: *Faculty of Computing, Blekinge Institute of Technology, Sweden*. 2020 (cit. on pp. 35, 36).

[23] *ESP32-DevKitC*. URL: https://www.espressif.com/en/products/devkits/esp32-devkitc (visited on 2024-01-10) (cit. on p. 49).

[24] *Exergame - Rehab and Physical Therapy*. URL: https://exergame.com/solutions/rehab-physical-therapy/ (visited on 2023-05-14) (cit. on p. 57).

[25] G. Gridling and B. Weiss. "Introduction to Microcontrollers". In: *Vienna University of Technology Institute of Computer Engineering Embedded Computing Systems Group*. 2007, pp. 1–8, 19, 25–30, 134–135 (cit. on p. 48).

[26] M. Gupton, A. Munjal, and R. R. Terreberry. *Anatomy, Hinge Joints*. PMID: 30085509. StatPearls, 2022 (cit. on p. 9).

[27] *HAMZAH, H. (2011). ACOUSTIC OBJECT LOCATING SYSTEM USING TRIANGULATION METHOD*. URL: https://utpedia.utp.edu.my/id/eprint/3257/1/Final_Report_Hazilah_11676.pdf (visited on 2023-10-10) (cit. on p. 19).

[28] *How do string pots work?* URL: https://www.te.com/usa-en/products/sensors/position-sensors/potentiometer-sensors/cable-actuated-position-sensors.html?tab=pgp-story (cit. on p. 50).

[29] *How does the knee work?* URL: https://www.ncbi.nlm.nih.gov/books/NBK561512/ (visited on 2022-10-29) (cit. on pp. 11, 12).

[30] *Inertial Measurement Unit (IMU) - An Introduction*. URL: https://www.advancednavigation.com/tech-articles/inertial-measurement-unit-imu-an-introduction/ (visited on 2024-01-08) (cit. on p. 19).

[31] *Internet of Things statistics for 2022 - Taking Things Apart*. URL: https://dataprot.net/statistics/iot-statistics/ (visited on 2023-02-01) (cit. on p. 41).

[32] *Introduction to SignalR*. URL: https://learn.microsoft.com/en-us/aspnet/signalr/overview/getting-started/introduction-to-signalr (cit. on p. 47).

[33] *Introduction to telehealth*. URL: https://www.physio-pedia.com/Introduction_to_Telehealth (visited on 2022-09-27) (cit. on p. 2).

[34] *IoT Communication Protocols—Network Protocols*. URL: https://www.allaboutcircuits.com/technical-articles/internet-of-communication-communication-protocols-network-protocols/ (cit. on p. 54).

[35] L. Ismail et al. "Requirements of Health Data Management Systems for Biomedical Care and Research: Scoping Review". In: *Journal of medical Internet research* 22 (2020-03). DOI: 10.2196/17508 (cit. on p. 30).

[36] R. G. J. Nagar A. Panat. "Knee Flexion/Extension Measurement System using Wearable Sensor". In: *International Journal for Research in Engineering Application and Management (IJREAM)* (2019). DOI: https://ijream.org/papers/IJREAMV05I0956024.pdf (cit. on p. 51).

[37] *Joint: Synovial*. URL: https://mydr.com.au/sports-fitness/joint-synovial/ (visited on 2022-11-07) (cit. on pp. 9, 10, 12).

[38] S. B. Khayani. *Development of Wearable Sensors for Body Joint Angle Measurement. Electronic Theses and Dissertations. University of Denver. 49.* 2011. URL: https://digitalcommons.du.edu/cgi/viewcontent.cgi?article=1048&context=etd (cit. on pp. 15, 24–26).

[39] J. N. Kishor das Thiago de Paula Oliveira. "Comparison of markerless and marker-based motion capture systems using 95 functional limits of agreement in a linear mixed-effects modelling framework". In: (2023-08). DOI: 10.1038/s41598-023-49360-2. URL: https://doi.org/10.1038/s41598-023-49360-2 (cit. on p. 16).

[40] *Knee Pain: Causes, Symptoms, Treatment and Exercises*. URL: https://rebalancetoronto.com/knee-pain-causes-symptoms-treatment-and-exercises/ (visited on 2022-09-27) (cit. on p. 14).

[41] S. Kobashi, Tsumori, et al. "Wearable knee kinematics monitoring system of MARG sensor and pressure sensor systems". In: *2009 IEEE International Conference on System of Systems Engineering (SoSE)*. 2009, pp. 1–6. URL: https://ieeexplore.ieee.org/document/5282337 (cit. on pp. 24, 25).

[42] S. Kong T. et al. "Physician attitudes towards-and adoption of-mobile health". In: *Digital health* (2020). DOI: https://doi.org/10.1177/2055207620907187 (cit. on pp. v, vi, 5).

[43] J. M. Lourenço. *The NOVAthesis LaTeX Template User's Manual*. NOVA University Lisbon. 2021. URL: https://github.com/joaomlourenco/novathesis/raw/master/template.pdf (cit. on p. ii).

[44] S. A. Marieswaran M. et al. "An extended OpenSim knee model for analysis of strains of connective tissues". In: *BioMed Eng OnLine* (2018). DOI: https://doi.org/10.1186/s12938-018-0474-8 (cit. on pp. 10, 13).

[45] R. Martínez et al. "Orientation Modeling Using Quaternions and Rational Trigonometry". In: *Machines* 10.9 (2022). ISSN: 2075-1702. DOI: 10.3390/machines10090749. URL: https://www.mdpi.com/2075-1702/10/9/749 (cit. on p. 22).

[46] E. M. H. Michael R Massoomi. "Increasing and Evolving Role of Smart Devices in Modern Medicine". In: (2019), pp. 181–186. DOI: https://doi.org/10.15420/ecr.2019.02 (cit. on p. 3).

[47] *Motion Capture Sensor Systems*. URL: https://www.azosensors.com/article.aspx?ArticleID=43 (visited on 2012-08-10) (cit. on p. 18).

[48] *Notifications overview*. URL: https://developer.android.com/develop/ui/views/notifications (cit. on p. 43).

[49] Y. Park, J. Lee, and J. Bae. "Development of a finger motion measurement system using linear potentiometers". In: *2014 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*. 2014, pp. 125–130. DOI: 10.1109/AIM.2014.6878066 (cit. on p. 18).

[50] *Physiotherapy in Kleinburg for Knee - Anatomy*. URL: https://www.advantagephysiotherapy.com/Injuries-Conditions/Knee/Knee-Anatomy/a~337/article.html (visited on 2022-11-07) (cit. on pp. 9, 11).

[51] G. Postolache et al. "IoT based model of healthcare for physiotherapy". In: *2019 13th International Conference on Sensing Technology (ICST)*. 2019, pp. 1–6. DOI: 10.1109/ICST46873.2019.9047710 (cit. on p. 4).

[52] *React - The Library for web and native user interfaces*. URL: https://react.dev/ (cit. on p. 33).

[53] *React Native - Core Components and Native Components*. URL: https://reactnative.dev/docs/intro-react-native-components (cit. on pp. 35, 37).

[54] *Relational vs. Non-Relational Databases*. URL: https://www.mongodb.com/resources/compare/relational-vs-non-relational-databases (cit. on p. 42).

[55] *Remote therapeutic monitoring*. URL: https://www.physitrack.com/remote-therapeutic-monitoring-rtm (visited on 2023-05-14) (cit. on p. 58).

[56] A. Reuter and M. Schindler. "Motion Capture Systems and Their Use in Educational Research: Insights from a Systematic Literature Review". In: *Education Sciences* 13 (2023-02), p. 167. DOI: 10.3390/educsci13020167. URL: https://www.mdpi.com/2227-7102/13/2/167 (cit. on p. 16).

[57] A. Reynolds, N. Awan, and P. Gallagher. "Physiotherapists' perspective of telehealth during the Covid-19 pandemic". In: *International Journal of Medical Informatics* 156 (2021), p. 104613. ISSN: 1386-5056. DOI: https://doi.org/10.1016/j.ijmedinf.2021.104613. URL: https://www.sciencedirect.com/science/article/pii/S1386505621002392 (cit. on pp. 1, 2).

[58] B. Rivera, Cano, et al. "A 3D-Printed Knee Wearable Goniometer with a Mobile-App Interface for Measuring Range of Motion and Monitoring Activities". In: *Sensors* 22.3 (2022). ISSN: 1424-8220. DOI: 10.3390/s22030763. URL: https://www.mdpi.com/1424-8220/22/3/763 (cit. on p. 28).

[59] *Rotation Quaternions, and How to Use Them*. URL: https://danceswithcode.net/engineeringnotes/quaternions/quaternions.html (cit. on pp. 23, 79, 80).

[60] *Rotations in Three-Dimensions: Euler Angles and Rotation Matrices*. URL: https://danceswithcode.net/engineeringnotes/rotations_in_3d/rotations_in_3d_part1.html (cit. on pp. 21, 22).

[61] F. Shokri et al. "The potential role of telemedicine in the infectious disease pandemic with an emphasis on COVID-19: A narrative review". en. In: *Health Sci. Rep.* 6.1 (2023-01), e1024 (cit. on p. 30).

[62] D. Silva. *System for Monitoring Physical Therapy Exercises - Interactive Mobile Application for Monitoring Device Measurements*. https://run.unl.pt/handle/10362/155570. 2023 (cit. on p. 129).

[63] *SomaticTelehealth*. URL: https://www.bluejayhealth.com/products/telehealth (visited on 2023-05-14) (cit. on p. 58).

[64] *System Design: Long polling, WebSockets, Server-Sent Events*. URL: https://dev.to/karanpratapsingh/system-design-long-polling-websockets-server-sent-events-sse-1hip (cit. on p. 44).

[65] *TDK InvenSense ICM-20948 - Datasheet*. URL: https://invensense.tdk.com/wp-content/uploads/2016/06/DS-000189-ICM-20948-v1.3.pdf (cit. on p. 23).

[66] *Telerehabilitation - What it is and Why It matters*. URL: https://www.globalmed.com/what-it-is-and-why-it-matters/ (visited on 2023-01-14) (cit. on p. 3).

[67] *The Anatomy of the Knee*. URL: https://www.physit.co.uk/the-anatomy-of-the-knee-by-melanie-white/ (visited on 2022-11-05) (cit. on p. 11).

[68] *The Best CSS Frameworks to Use in Your Projects for 2024 and Beyond*. URL: https://www.freecodecamp.org/news/best-css-frameworks-for-frontend-devs/#:~:text=Component%2Dbased%20frameworks%20offer%20a,starting%20from%20scratch%20every%20time. (cit. on p. 34).

[69] *The Clean Architecture*. URL: https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html (cit. on p. 39).

[70] J. Tosi et al. "Performance Evaluation of Bluetooth Low Energy: A Systematic Review". In: *Sensors* 17 (2017-12), p. 2898. DOI: 10.3390/s17122898 (cit. on pp. 55, 56).

[71] *TypeScript - What is Typescript*. URL: https://www.typescriptlang.org/ (cit. on p. 33).

[72] S. Uhlrich et al. "OpenCap: Human movement dynamics from smartphone videos". In: *PLOS Computational Biology* 19 (2023-10), e1011462. DOI: 10.1371/journal.pcbi.1011462 (cit. on p. 17).

[73] P. Venkataanusha et al. "Detecting Outliers in High Dimensional Data Sets Using Z-Score Methodology". In: *International Journal of Innovative Technology and Exploring Engineering* 9 (2019-11), pp. 48–53. DOI: 10.35940/ijitee.A3910.119119 (cit. on p. 110).

[74] *Virtual DOM and Internals*. URL: https://legacy.reactjs.org/docs/faq-internals.html (cit. on pp. 33, 34).

[75] *Web app vs mobile app: which is better for your healthcare product?* URL: https://www.twentyideas.com/blog/health-web-app-vs-mobile-app (cit. on pp. 31, 32).

[76] *What is a Flex Sensor and How to use it with Arduino*. URL: https://circuitdigest.com/microcontroller-projects/interfacing-flex-sensor-with-arduino#:~:text=A%20flex%20sensor%20is%20a%20kind%20of%20sensor%20which%20is,sensor's%20resistance%20will%20be%20changed. (visited on 2024-03-05) (cit. on p. 50).

[77] *What Is Cloud Computing? Definition, Benefits, Types, and Trends*. URL: https://www.spiceworks.com/tech/cloud/articles/what-is-cloud-computing/ (visited on 2022-02-01) (cit. on p. 41).

[78] *What is mHealth?* URL: https://www.techtarget.com/searchhealthit/definition/mHealth (visited on 2022-09-27) (cit. on p. 3).

[79] *What is optical motion capture?* URL: https://www.motionanalysis.com/biomechanics/what-is-optical-motion-capture/ (visited on 2023-10-29) (cit. on p. 16).

[80] *What Is Range of Motion?* URL: https://www.verywellhealth.com/overview-range-of-motion-2696650 (visited on 2022-11-22) (cit. on pp. 13, 14).

[81] *What's the Difference Between an ACID and a BASE Database?* URL: https://aws.amazon.com/compare/the-difference-between-acid-and-base-database/ (cit. on p. 42).

I

Annex

I.1 Evaluation - Patient Questionnaire

1. Was the navigation to the appointment booking view intuitive and concise?
2. Regarding the user interface of appointment booking, did you find the dropdown component in conjunction with the calendar component fitting for presenting the physiotherapist's appointment availabilities?
3. Was the calendar component confusing for visualizing the physiotherapist appointment availabilities?
4. Did you find booking an appointment an easy task?
5. Did you find the layout of the protocol view confusing?
6. Regarding the user interface of a protocol, did you find the exercise media useful for conducting the physiotherapy exercise in a well formed manner?
7. Did you find the calibration phase easy to follow and also did you consider the pairing process between the prototype and the mobile application self-explanatory?
8. Did you find the wearable sensor prototype comfortable and easy to apply?
9. Did you find the wearable sensor prototype actuators useful when conducting a physiotherapy exercise?
10. Regarding the user interface of a exercise, did you like the gauge component as form of representing the upper and lower boundaries regarding the knee range of motion as well as present the measured angle?
11. In the exercise screen, apart from the available metrics what would you add?
12. Regarding the patient application what improvements would you suggest?

I.2 Evaluation - Physiotherapist Questionnaire

1. During the exercise type creation, was it an intuitive task when selecting the multimedia content to be associated to the exercise?
2. Did you find the protocol creation process too overwhelming and should it be segregated in smaller steps?
3. During the protocol creation process, did you find the exercise parameterization a useful feature for creating different exercises?
4. During the protocol creation process, did you find the exercise scheduling limiting?
5. When performing the confirmation of a pending appointment, was it intuitive?
6. Regarding the chat creation upon confirming an appointment, do you think chats should only be created by through this mean or should an alternative way be provided?
7. Regarding notifications, did you find them useful for providing engagement?
8. Regarding notifications, did you find them unnecessary and annoying?
9. Did you find the reports insightful for detecting knee range of motion improvement or deterioration?
10. Do you find different file formats both patient data and appointment data useful?
11. What improvements could be made in the reports?
12. What improvements could be made in the physiotherapist application?