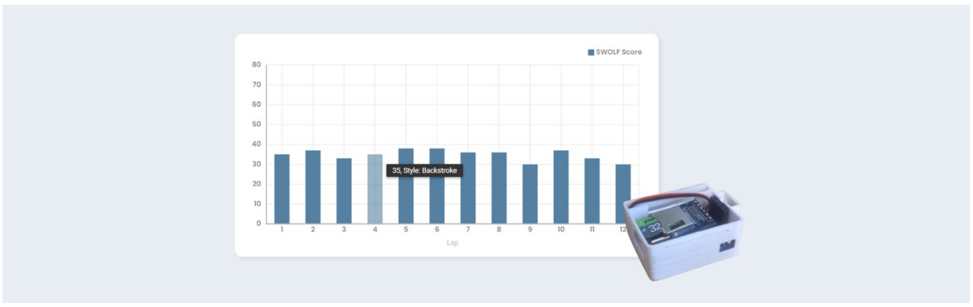
Instituto Superior de Engenharia de Lisboa (ISEL)

Departamento de Engenharia Eletrónica e de Telecomunicações e Computadores (DEETC)

LEIM
Licenciatura em Engenharia Informática e Multimédia
Unidade Curricular de Projeto

Swimmers' Training Log Application



Daniela Filipa Levezinho da Silva (A47472)
Letícia Inês da Silva Barbedo Lucas (A48584)

Advisors

Professor Carlos Gonçalves – Instituto Superior de Engenharia de Lisboa
Professor Pedro Teodoro – Escola Superior Náutica Infante D. Henrique

Lisbon, July 2023

Abstract

This work expands upon the final project conducted by Gabriel Diaz, a master's student, entitled "Machine Learning of Swimming Styles,"more commonly simply referred to as "SWIMU". Ultimately, the project aimed to build a low-cost wearable device with open-source code capable of being used in swimming practice.

This project consisted of an implementation of prototype wristbands, capable of collecting, storing, and analyzing angle data during swimming sessions to provide valuable insights and enhance athletes' performance. With detection accuracy rates consistently around 95% for different swimming styles, the original project opened doors for a cheaper and more accessible option compared to existing swimming devices on the market, which tend to be more expensive due to the need to be water-resistant.

While focusing primarily on swimmers, the original project strives to create a versatile and user-friendly sports wristband solution that could be utilized by athletes from diverse disciplines.

In the current project, the goal was to further advance the original work by creating an application - another piece for the system. This application should not only allow the monitoring of training sessions but also retrieve the data stored in the sensor module, enabling a more personalized record and display of a swimmer's workout. This includes information such as the distance swum, the number of different swimming styles practiced, the time taken for each style, and an overall assessment of the quality of the swimmer's training, incorporating specific metrics for swimming performance, such as SWOLF measurements.

The wristband will accurately measure angles during workouts and store the acquired data on a memory card. Additionally, the data will be transmitted to a dedicated web application via the serial port for further analysis.

To ensure accessibility, the analyzed statistics will be stored in a Cloud service. This allows athletes to access their training data and statistics from any device with a web browser, empowering them to make informed decisions and optimize their training sessions.

While initially targeting stakeholders from the Escola Náutica Infante D. Henrique, the ultimate objective is to generalize the use of this wristband to benefit athletes from various sports disciplines. This inclusivity will enable a broader audience to leverage the advantages of this training tool, promoting performance improvement and achieving training goals.

Keywords: Inertial Measurement Unit (IMU); Prototype Wristbands; Web Application; Mobile Application; Performance Improvement; Swimming Styles; Data Analysis; Open-source; Lowcost; Dashboards.

Resumo

Este trabalho expande o projeto final realizado pelo Gabriel Diaz, um estudante de mestrado, intitulado “Aprendizagem Automática de Estilos de Natação”, mais vezes referido como “SWIMU”. Dando uma vista geral, o projeto teve como objetivo construir um dispositivo *wearable* de baixo custo com código *open-source* que possa ser usado em treinos de natação.

Passou pela criação de pulseiras protótipo capazes de obter, armazenar e analisar dados relativos a ângulos de movimento durante sessões de natação para fornecer informações pertinentes sobre a técnica a fim de aprimorar o desempenho dos atletas. Com probabilidades de acerto consistentemente a rondar os 95% para os diversos estilos de natação, o projeto original abriu portas a uma opção mais barata e acessível em comparação com os dispositivos de natação existentes atualmente no mercado, que tendem a ser mais caros devido à necessidade de poderem ser usados dentro de água.

Embora o foco fosse principalmente nadadores, o objetivo do projeto original foi criar uma solução de pulseiras versátil e fácil de usar que pudesse ser utilizada por atletas de diversas disciplinas.

No projeto atual, o objetivo foi avançar ainda mais o trabalho original, com a criação de uma aplicação - mais uma peça do sistema. Esta aplicação deverá não só permitir monitorizar as sessões de treino, mas também obter os dados armazenados no módulo do sensor, permitindo mostrar os mesmos de umas forma mais personalizada. Isto inclui informações como a distância nadada, o número de estilos de natação praticados, o tempo gasto em cada estilo e uma avaliação geral da qualidade do treino do nadador, incorporando métricas específicas para o desempenho na natação, como a medida SWOLF.

A pulseira irá medir com precisão os ângulos durante os treinos e armazenar os dados adquiridos num cartão de memória. Os dados serão então transmitidos para uma aplicação web dedicado por meio da porta serial para análise.

Para garantir o acesso, as estatísticas analisadas serão armazenadas num serviço *Cloud*. Isto permite que os atletas possam aceder aos seus dados de treino a partir qualquer dispositivo com um *web browser*, capacitando-os a tomar decisões informadas e otimizar suas sessões de treino.

Embora inicialmente direcionado aos interessados da Escola Náutica Infante D. Henrique, o objetivo final é generalizar o uso deste dispositivo para beneficiar atletas de várias disciplinas desportivas. Esta inclusividade permitirá que um público mais largo possa aproveitar as vantagens desta ferramenta de treino, promovendo a melhoria do desempenho e o alcance das suas metas de treino.

Palavras-Chave: Inertial Measurement Unit (IMU); Pulseiras Protótipo; Aplicação *Web*; Aplicação Móvel; Melhoria de Desempenho; Estilos de Natação; Análise de Dados; Código Aberto; Baixo Custo; *Dashboards.*

Index

Abstract **i**

Resumo **iii**

Index **v**

List of Tables **ix**

List of Figures **xi**

List of Listings **xiii**

List of Acronyms **xv**

1 Introduction **1**

1.1 Contribuitions 2

1.2 Report Organization 4

2 Related Work **5**

3 Proposed Model **9**

3.1 Requirements 9

3.1.1 Functional Requirements 9

3.1.2 Non-Functional Requirements 12

3.2 Use Cases 13

3.3 Foundations 15

3.4 Proposed Architecture 17

4 Model Implementation 19
4.1 Sensor Module 19
4.1.1 Seeed Studio XIAO nRF52840 (Sense) 20
4.1.2 Cost of the Project 21
4.1.3 Inertial Measurement Unit - IMU 22
4.1.4 Bluetooth Low Energy Characteristics 24
4.1.5 Session Files 26
4.1.6 Data Transfer - USB or Bluetooth? 29
4.1.7 Sensor State-Machine 31
4.2 Mobile Application 35
4.2.1 Setting up React Native 35
4.2.2 Implementation 36
4.3 Web Application 38
4.3.1 Structure and Communication 39
4.3.2 Server Setup 41
4.3.3 File Transfer 43
4.3.4 Cloud Service 45
4.3.5 Client 47
4.3.6 Data Presentation 48
4.3.7 Data Treatment and Calculations 52
4.3.8 Style Identification 57

5 Validation and Testing 61
5.1 Validation of the Angle Values Obtainment 61
5.2 Mobile Connection and Communication 64
5.3 Stroke Calculation 65
5.4 Full Session Testing 68
5.4.1 Same Style Every Lap 68
5.4.2 Varying Styles 70
5.5 Session Comparison 72

6 Conclusions and Future Work 73
6.1 Future Work 74

A Board and Circuit Details 77

B Web Application Schema **79**

References **81**

List of Tables

2.1 Swimming devices features 6
2.2 Project features 7

3.1 Functional Requirements - Sensor Module 10
3.2 Functional Requirements - Mobile Application 10
3.3 Functional Requirements - Web Application 11
3.4 Non-functional Requirements 12

4.1 Estimated project cost. 21
4.2 Bits allocation in a 2 hour file. 28
4.3 Study in total file size. 29

5.1 Study of the results for 50 strokes - freestyle. 66
5.2 Study of the results for varying number of strokes - freestyle. . 67
5.3 Study of the number of strokes variation. 67
5.4 Test Session 1. 68
5.5 Test Session 1 results. 69
5.6 Test Session 2. 70

List of Figures

3.1 First use case scenario. 13
3.2 Second use case scenario. 14
3.3 Swimming styles examplification 15
3.4 Project architecture. 17

4.1 Sensor components and axis. 20
4.2 Yaw, pitch and roll as seen on the sensor itself. 23
4.3 Bluetooth BLE hierarchy. 25
4.4 Sensor components and axis. 30
4.5 Sensor state-machine. 32
4.6 Communication protocol. 34
4.7 First page of the mobile application. 36
4.8 Second page of the mobile application. 38
4.9 Client-server communication via HTTP. 40
4.10 Javascript module in charge of serial utilities. 44
4.11 Javascript module in charge of Cloud service utilities. 45
4.12 Swimming journey page. 49
4.13 Session page. 49
4.14 Hovering over a value on the SWOLF graph. 50
4.15 Bar chart variant. 51
4.16 Javascript module in charge of data treatment utilities. 53
4.17 Roll signal obtained for a swimming session. 53
4.18 Example of maximas in a portion of the session. 55
4.19 Addition of minimas in the same portion of the session. 56
4.20 Comparison of pitch and roll for freestyle and breaststroke. . . 58

5.1 Roll values obtained. 62

5.2 Pitch and yaw values obtained. 62
5.3 Freestyle “stages”detected. 63
5.4 Mobile bluetooth screen detecting a device. 64
5.5 Lights changing depending on the sensor module’s state. . . . 65
5.6 Results of the first session. 69
5.7 Results of the second session. 71
5.8 Functioning comparison page. 72

A.1 Sensor Schematic. 77
A.2 Printed Circuit Board - Top View. 78
A.3 Printed Circuit Board - Bottom View. 78

B.1 Server schema. 79
B.2 Client schema. 80

List of Listings

1 Content of the session file 27
2 Server setup example 42
3 Function to find local maximas 54

List of Acronyms

API	Application Programming Interface.
BCE	Before the current era.
BLE	Bluetooth Low Energy.
CSV	Comma Separated Values.
CTS	Current Time Service.
GATT	Generic Attribute Profile.
HTML	Hypertext Markup Language.
HTTP	Hypertext Transfer Protocol.
IMU	Inertial Measurement Unit.
ISEL	Instituto Superior de Engenharia de Lisboa.
JSON	JavaScript Object Notation.
MEIM	Mestrado em Engenharia Informática e Multimédia.
SD	Secure Digital.
SVG	Scalable Vector Graphics.
UML	Unified Modeling Language.
URI	Unique Resource Identifier.

URL	Uniform Resource Locators.
USB	Universal Serial Bus.
UUID	Universal Unique Identifier.

Chapter 1

Introduction

Swimming, an age-old practice that has transcended time, has evolved into both an art form and a competitive sport. From the earliest recorded instances of humans navigating through water to the highly refined techniques seen in modern swimming competitions, the evolution of swimming is a testament to the remarkable adaptability and ingenuity of our species. Over centuries, swimming has not only undergone significant changes in terms of form and style but has also developed into a means of physical fitness, leisure, and even a form of therapy. The journey of swimming's evolution is a fascinating exploration of human progress and our unyielding desire to conquer the water.

In ancient times, swimming was primarily a survival skill, enabling humans to navigate bodies of water for fishing, transportation, and avoiding danger. However, as civilizations flourished and exploration expanded, swimming gradually evolved into a recreational activity. It became a source of pleasure, with early depictions of swimming found in ancient Egyptian hieroglyphics and Greek art. As societies progressed, swimming began to take on a more competitive nature, with the first organized swimming competitions recorded in Japan during the 1^{st} century BCE (Before the current era).

The transformative phase of swimming occurred during the 19^{th} and 20^{th} centuries, marked by significant advancements in technique, equipment, and the establishment of swimming as an Olympic sport. Pioneers like Captain Matthew Webb, who became the first person to swim across the English Channel in 1875, and the introduction of the butterfly stroke in the early 20^{th}

century, propelled swimming into new realms of achievement. Today, swimming continues to captivate enthusiasts and athletes alike, with cutting-edge training methods, state-of-the-art facilities, and record-breaking performances pushing the boundaries of human capability.

Last year, Gabriel Diaz, a master's student, developed a project entitled "Machine Learning of Swimming Styles"more commonly simply referred to as "SWIMU"[Diaz, 2022]. It consisted of an implementation of prototype wristbands, capable of collecting, storing, and analyzing angle data during swimming sessions to provide valuable insights and enhance athletes' performance. This process made use of Machine Learning Algorithms and resulted in an extremely promising solution, with detection accuracy rates consistently around 95% for various swimming styles, with an algorithm capable of being adapted to other sports.

This project opened doors for a cheaper and more accessible option compared to existing swimming devices on the market, which tend to be much more expensive due to the need to be water-resistant.

The current project aims to further advance the original work by adding a new piece to the system, one that's a little more user oriented: A mobile and web application.

The applications should not only allow the monitoring of training sessions, by starting and ending sessions via communication with the sensor module, but also the retrieval of the data stored in this module. It will then allow users to view a more personalized record of their workout. This includes information such as the distance swum, the average time per lap, the number of laps, the swum distance and even permit the comparison between training sessions. It also strives to incorporate specific metrics for swimming performance, such as SWOLF measurements.

1.1 Contribuitions

At present time, there is a very wide range of wearables that can be used in the most varied situations: this includes smartwatches and/or smartbands for sports activities. These wearables allow, for example, counting the number of steps, calories burned, and specific metrics for different types of sports and much more. For devices that focus primarily on swimming, these metrics often include stroke count, time and distance swum, differentiation of swimming styles, among others metrics. However, the overwhelming majority of these devices are proprietary and do not allow free access to the collected data and, due to the particular need of being water proof, they tend to be relatively more expensive. A study on the state of the art will be conducted in Chapter 2.

This project provides a solution to this matter, as it presents a open-source and low-cost solution applied to the swimming practice.

At the beginning of the project, two components were passed on from Diaz's project: the retrieval of inertial data and the swimming style reference database. The developed system, described in thsi report, includes the following scientific and technical contributions:

- Development of software for a Seeed Studio XIAO nRF52840 (Sense) microcontroller, part of the sensor module, capable of collecting and saving data during swimming sessions in a Secure Digital (SD) card;
- Development of communication protocols used to start and end training sessions as well as to obtain the swimming data via external devices;
- Design and development of a mobile application, available for both Android and iOS devices, capable of starting and ending training sessions, and sending to the sensor module basic configurations important for data analysis;
- Design and development of a web application, capable of retrieving the session data from the sensor module and of storing it in a Cloud storage service;
- This web application is also capable of both displaying a overview of relevant swimming statistics such as swum distance, number of strokes, number of laps, swum styles and SWOLF score via dashboards and also showing the evolution of the swimming via session comparison;
- Analysis system capable of performing the aforementioned calculations;

Furthermore, the project's dossier is available can be found in the report's references, under [Letícia Lucas, Daniela Levezinho, 2023].

1.2 Report Organization

Besides the current introduction chapter, that has just been concluded, this report is organized as follows.

Chapter 2 delves into a study of published works whose theme and ideas resemble what will be described in this document, these works are characterized and compared, with the study of their distinctive aspects.

The identification of functional and non-functional requirements, as well as use case scenarios will then be carried out in Chapter 3. As a means to give the reader a better understanding of the choices implemented, the foundational principles that support the work will also be presented, along with a brief analysis of the architecture followed.

Chapter 4 will thus expand upon that architecture, formally explaining each component in detail.

The description of the tests and validations are presented in Chapter 5, as a way to verify the overall functioning of the project, considering the proposed objectives and usage scenarios previously mentioned.

Finally, Chapter 6 concludes with a summary of the application development process, along with a couple ideas and considerations for future work that may be carried out.

To conclude, Appendix A includes the sensor schematic as well as the printed circuit board while Appendix B contains a schematic regarding the web application classes.

Chapter 2

Related Work

In this chapter, an analysis of other works and applications that exist in the market will be briefly presented. All of these share a common objective similar to the work presented in this report.

One noteworthy project is Gabriel Diaz's final master's project, titled "Machine Learning of Swimming Styles - SWIMU"[Diaz, 2022] Diaz's research resulted in the development of an advanced and meticulously trained model that is specifically designed to distinguish between different swimming styles. The developed model provides highly accurate and dependable values, enabling precise classification and differentiation of various types of swimming techniques.

In addition to Diaz's project, it is worth noting that there are several commercial brands that offer watches and wristbands designed to collect training data, some of which are tailored specifically for swimming. These products claim to facilitate the analysis of a swimmer's performance.

Solutions such as Garmin [Garmin, 2023a], Fitbit [Fitbit, 2023b], and Apple Watch [Apple, 2023b] emerge among the devices available in the market for swimming. The features of each device is shown below in Table 2.1.

Table 2.1: Swimming devices features

Devices	**Garmin**	**FitBit**	**AppleWatch**
Detect Swim Styles	YES	YES	YES
Tracks Distance	YES	YES	YES
Accuracy	YES	NO	YES
Proprietary	YES	YES	YES
Training Plan	YES	NO	YES
Price	up to €1000	up to €200	up to €1000

Starting with the brand Garmin [Garmin, 2023b], they produce watches for sports in a very generalized manner. The first noticeable aspect of their watches is the high price, typically ranging from €400 to €1000. Additionally, they are physically quite large, which can hinder some movements during swimming and may even become uncomfortable for individuals with smaller wrists, which is common among females. Despite this, they offer various features expected for analyzing swimming performance, and reviews are generally positive in terms of accuracy in tracking distances swum and swimming styles.

Fitbit [Fitbit, 2023a], on the other hand, is much more affordable, ranging between €100 and €200, but it receives many criticisms regarding inconsistent data and errors in both the recognition of swimming styles and the accurate measurement of distances covered. Therefore, it is recommended for more casual swimmers.

Apple Watches [Apple, 2023a] have a higher price range, between €300 and €1000, and a relatively shorter battery life compared to the others. However, they provide all the necessary statistics for a swimming watch, including training planning and detection of changes in swimming style mid-lap, appearing to have the highest level of accuracy among all the other watches.

It is relevant to mention that none of these devices allow for the identification and correction of swimming errors and also none of them are open-source, meaning it's not possible to modify aspects of the implementation or to add new features.

This project should also be further compared to the works referred by being analyzed in the same light. Table 2.2 shows this study.

Table 2.2: Project features

Device	**SWIMU**
Detect Swim Styles	Using Diaz's study, determination of style will be possible
Tracks Distance	Estimates distance based on laps swum and pool size
Accuracy	Will be studied in Chapter 5
Proprietary	Open-source, as is Diaz's work
Training Plan	No

The strengths of this project will be further highlighted as the implementation process is explained and the results validated. For now, one of the key factors that can be focused on is the fact that the project's code is open source. This means that all the features that won't be fulfilled with this the current project, could still be implemented later down the lines: the final state of SWIMU is not set in stone.

One of the main takeaways from this project is just how diverse the possibilities from here on out are. There's a lot of room for growth and innovation in the future.

Chapter 3

Proposed Model

This chapter presents a comprehensive exploration of the proposed model, explaining the proposed solution to address the problem at hand. It begins with an overview of the functional requirements (Section 3.1) that drive the model's development, followed by the identification of use-cases (Section 3.2). Subsequently, key elements and notions (Section 3.3) are introduced to provide a better understanding of the model's implementation. The chapter concludes with an explanation of the proposed architecture (Section 3.4), highlighting its structure and anticipated benefits. Together, these sections lay the foundation for a thorough understanding of the proposed model and its potential impact.

3.1 Requirements

To review, the primary objective of this project is to develop a sports wristband that enables the collection and visualization of crucial data and statistics regarding an athlete's training directly on an application, via the use of dashboard. Initially targeting swimmers of various abilities, it aims to create a solution that can be easily adapted to other sports in future endeavors. This section will describe both the functional and non-functional requirements, followed by the exemplification of two use cases.

3.1.1 Functional Requirements

Functional requirements define the specific functionalities and features that a system or project must possess to fulfill its intended purpose. They describe what the system should do and how it should behave in response to various inputs or events. As the project implementation is divided into various components these requirements will also be presented in separate for better comprehension.

Sensor Module Table 3.1 shows the list of the functional requirements regarding the sensor module.

Table 3.1: Functional Requirements - Sensor Module

Requirement	**Description**
Data Acquisition	Acquire data effectively during training sessions, ensuring an adequate number of samples without data loss.
Session Saving	Enable the saving of multiple sessions per day by determining the most suitable method for data storage.
Calibration	Implement sensor calibration over time to compensate for any drifting.
Data Transmission	Transmit data to the web application for storage and analysis.

These requirements then refer to the initial data obtainment storage and transmission in the sensor module. Ensuring these requirements in fundamental, as they lay the foundation of the remaining functionalities.

Mobile Application In Table 3.2 the list of the functional requirements regarding the mobile application can be seen.

Table 3.2: Functional Requirements - Mobile Application

Requirement	**Description**
Connection to the Sensor	Establish a Bluetooth BLE connection to the sensor module.
Handling Sessions	Prompt the sensor to start or stop saving session data by sending necessary parameters.

By fulfilling these requirements, the mobile application will be able to connect to the sensor module and prompt it to begin its functions.

Web Application Lastly, Table 3.3 lists functional requirements regarding the web application.

Table 3.3: Functional Requirements - Web Application

Requirement	**Description**
Data Reception	Receive data from the sensor module for storage and analysis.
Data Storage	Store transmitted data of each session in a way that's easily accessed by the web application.
Stroke Identification	Identify the number of strokes in a session.
Lap Identification	Identify the number of laps in a session.
Distance Identification	Identify the total distance swum in a session.
Style Identification	Identify the swimming styles used in each lap.
SWOLF Score Calculation	Calculate the SWOLF score per lap, a measure of swimming efficiency.
Dashboard Display	Display relevant information in a dashboard for each session, including the results of the previously mentioned calculations.
Progress Visualization	Provide visual representation of progress across multiple training sessions.
Session Deletion	Allow users to delete sessions from the server.
Session Comparison	Enable comparison of training sessions for performance analysis.

These functional requirements aim to permit the presentation of session data to the users.

In conclusion, the identified functional requirements play a crucial role in ensuring the success and effectiveness of the project. From establishing a seamless connection with the sensor module and acquiring data efficiently to providing comprehensive analysis and visualization of swimming sessions, these requirements address key aspects of the system's functionality.

3.1.2 Non-Functional Requirements

Non-functional requirements, describe the qualities or characteristics of a system, rather than its specific functionalities. They define the constraints and criteria that the system should meet in terms of its performance, usability, reliability, security, and other attributes. Thus, a set of non-functional requirements was identified, as presented in Table 3.4.

Table 3.4: Non-functional Requirements

Requirement	**Description**
Reliability	The system should perform consistently without errors or failures.
Performance	Acquire and process data as efficiently as possible to provide real-time or near-real-time feedback.
Scalability	Handle a growing number of users and data without significant performance degradation.
Usability	Provide a user-friendly interface that is easy to navigate and interact with.
Compatibility	Support a wide range of devices and operating systems for maximum accessibility.
Maintainability	Design the system in a modular and well-structured manner to facilitate future updates and maintenance.
Accuracy	Provide accurate measurements and analysis of swimming-related metrics.
Compliance	Adhere to industry standards and regulations for data security and privacy.

In conclusion, the non-functional requirements drive key aspects of the project, ensuring essential qualities such as security, accuracy, performance, scalability, and others.

By focusing on the aforementioned functional and non-functional requirements, the development of the sports wristband and accompanying application aims to enhance the training experience and performance evaluation for athletes, with potential for broader application in various sports in the future.

3.2 Use Cases

It's now important to further expand upon these requirements by analysing possible use cases. Such cases can be separated in two different stages of interaction. The first one refers to the interaction with the mobile application, as the user starts and ends a training session and the second one focuses on the interaction with the web application, as the users transfer their session data and in order to view swimming statistics.

Mobile and Sensor Let's start with the characterization of the scenario involving the use of the mobile application and sensor module to deploy, conduct and finish a training session. Figure 3.1 shows this first scenario.

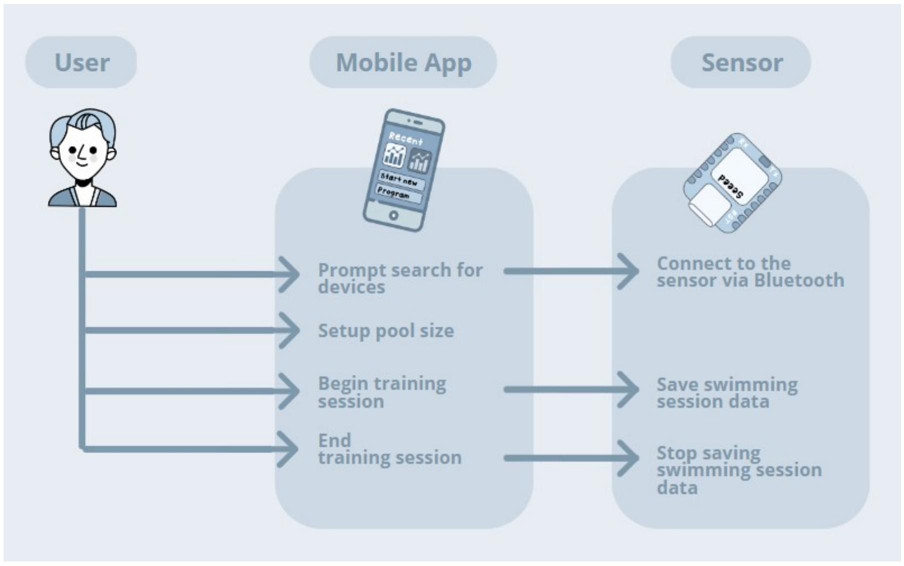

User
Mobile App
Sensor
Prompt search for devices
Setup pool size
Begin training session
End training session
Connect to the sensor via Bluetooth
Save swimming session data
Stop saving swimming session data

Figure 3.1: First use case scenario.

To sum it up, users will be able to launch the mobile application and promptly search for devices to connect to. A device must then be picked from the given list - this should be the SWIMU device that will be used.

Then, users will be able to configure the pool size they'll be swimming in - this is necessary to properly calculate the swum distance later down the line - and prompt the beginning of a training session. The connected device will immediately start fetching and saving data until the session is stopped by the user via the mobile application, as well.

Web The second case refers to the visualization of training statistics on the web application, after proper transfer. It follows as shown in Figure 3.2.

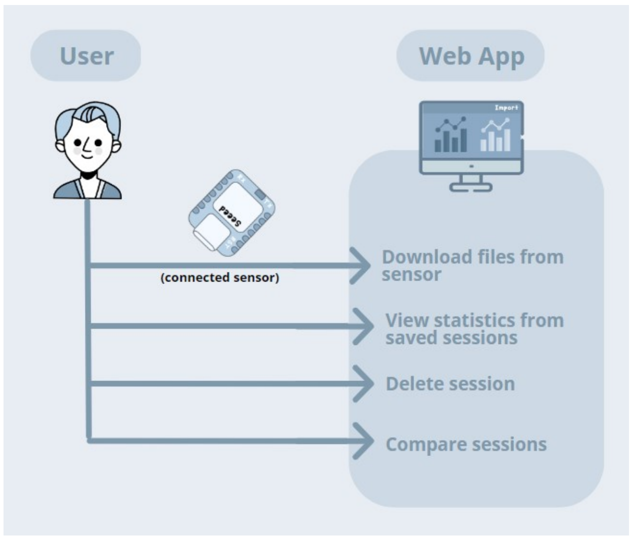

User
Web App
(connected sensor)
Download files from sensor
View statistics from saved sessions
Delete session
Compare sessions

Figure 3.2: Second use case scenario.

This case now shows the interaction with the web application. The user can view the details of training sessions (SWOLF calculations, stroke count, lap count, style identification, and more.) as well as compare their sessions as a way to see the progress. The sessions are uploaded in the background as soon as the user connects the USB cable, establishing a connection between the device running the application and the microcontroller used, and promptly saved and treated to be shown as previously described.

The user can also delete sessions from the server should the results have, for example, suffered any anomalies.

3.3 Foundations

This section aims to expand some notions that should be acquired to better understand the implementation of this project and the choices made within it.

As this project is expected to primarily be used by swimmers, and all calculations and dashboards are thus directed at this sport, it's important to learn a little about the sport of swimming and its specific nomenclatures in terms of styles, and performance metrics. This knowledge enabled the group to design and implement features that cater to the unique needs of swimmers.

Swimming is a popular recreational and competitive sport that involves propelling oneself through water using various techniques. There are four main swimming styles, also known as strokes: freestyle, backstroke, breaststroke, and butterfly. Each style has its own distinct technique and rules. Figure 3.3 shows a visual representation of each style.

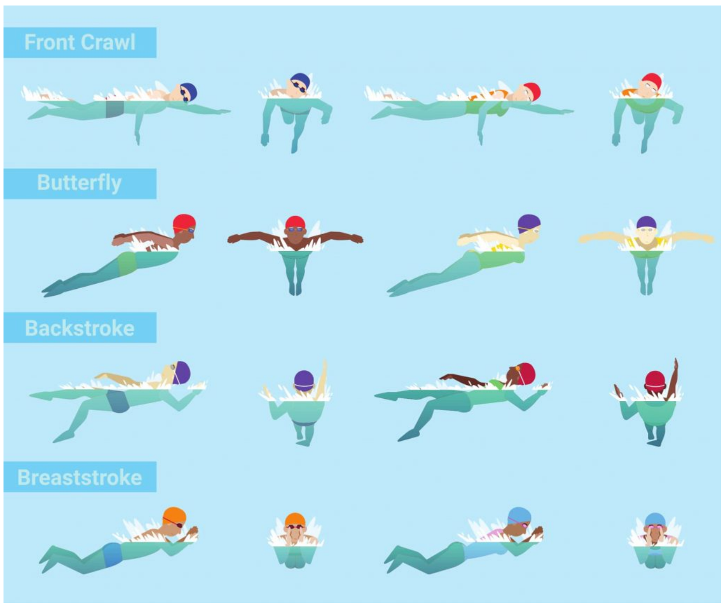


Figure 3.3: Swimming styles examplification - from 'Little Fishies' [Little Fishies, 2021].

Freestyle, also referred to as front crawl, is the fastest and most commonly used swimming style. Swimmers use alternating arm movements and a flutter kick to propel themselves through the water. Breathing is typically done by turning the head to the side during arm recovery, allowing regular breaths without interrupting the athlete's rhythm.

Similar in movement, backstroke is performed on the back. Swimmer use an alternating motion of their arms while kicking their legs in a flutter kick. The face remains above the water, and breathing is done naturally as the swimmer rotates their head to the side. Its relaxed and comfortable nature, as swimmers have a clear view of the sky or ceiling while swimming.

Breaststroke is the oldest known swimming style and is characterized by a simultaneous movement of both arms from a stretched-out position to the chest, followed by a powerful kick. Swimmers glide between arm strokes and take a breath while lifting their head out of the water. Compared to freestyle or butterfly, it flows at a slower pace.

Finally, there's the butterfly style. It's often considered the most challenging and demanding of the four strokes, involving a simultaneous movement of both arms in a circular pattern, while the legs perform a dolphin kick. Swimmers maintain a continuous butterfly motion throughout the race, with no underwater pulls or kicks allowed after the start or turns.

Each swimming style has its own unique set of challenges and requires specific techniques to excel. And as such, with such varied arm movements, the data obtained from the sensor module will be distinct enough to allow for style identification (the data type will be expanded upon in Section 4.1.3 and the calculations will be explained in Section 4.3.8).

This report has mentioned the SWOLF metric a few of times, let's now properly explain what it means and why it's so important.

The SWOLF metric is a measurement used in swimming as a means to assess an athlete's technique efficiency and effectiveness. SWOLF combines the words 'Swim' and 'Golf' creating a comparative metric similar to that of golf's stroke count. It provides a way to gauge the efficiency of each stroke taken during a swimming lap.

To calculate SWOLF, the swimmer adds their time for one pool length (in seconds) to the number of strokes it took to complete that length. For example, if a swimmer completes a length in 30 seconds and takes 20 strokes, their SWOLF score for that length would be 50. The lower the SWOLF score, the more efficient the swimmer is considered to be.

The SWOLF metric aims to encourage swimmers to find the optimal balance between speed and stroke efficiency. It promotes the idea that a swimmer can improve their overall performance not only by increasing their speed but also by reducing the number of strokes taken. By focusing on stroke technique and efficiency, swimmers can strive to achieve a lower SWOLF score, indicating that they are using their energy more efficiently.

This metric can then be a very helpful tool for swimmers and coaches to track progress and identify areas for improvement - it will be the metric this project's dashboards will primarily focus on.

3.4 Proposed Architecture

This chapter will expose the architecture to fulfill the projects goals. It aims to also provide a first layer of justifications for the choices taken. Figure 3.4 shows the proposal.

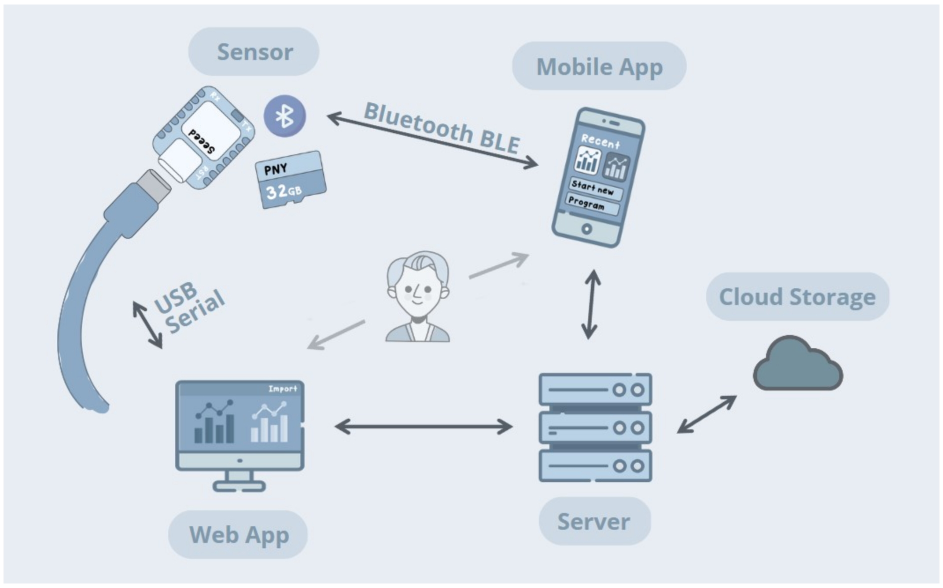


Figure 3.4: Project architecture.

To give a brief overview, as all the components will be further expanded upon in the implementation chapter (refer to Chapter 4), let's go through each of the modules.

The sensor module is explained in detail starting in Section 4.1. It's the most fundamental block of the architecture. It's in charge of not only collecting and saving data from swimming sessions when prompted by the mobile app via Bluetooth BLE, but also of transferring said data to the web application via Serial communication.

One of the components the sensor module communicates with is the mobile application, implemented using React Native as a way to provide support to both Android and iOS. It connects to said module via Bluetooth BLE and sends messages regarding the beginning, end and configuration of training sessions. See Section 4.2.

Regarding the web application, its the component that fulfills the goal of showing the user the information received from the sensor module in an interactive and user-friendly way, via a dashboard. See Section 4.3, where this module is expanded.

The Server is then described in Section 4.3.2, it receives requests from the browser, processes these requests and performs any necessary computations or data retrieval, promptly returning the results back to the client.

In Section 4.3.4, the Cloud storage component is also described - this refers to the service where all the files transferred from the sensor module are saved. Regardless of the platform users choose to deploy the web application in, it will allow an easy access to their saved sessions.

Chapter 4

Model Implementation

The chapter describes the implementation of the model architecture, highlighting its structure and development layers.

It begins by explaining the sensor module in Section 4.1, with the description of the data collecting logic and the communication protocol, both with the mobile and web application. Starting on Section 4.2 the mobile application component will be further presented. The last component, the web application, is expanded upon in Section 4.3, with the data analysis process being delineated in that same section. The challenges and limitations encountered during the implementation process will also be discussed throughout each section.

4.1 Sensor Module

This section provides insights into the hardware implementation, specifically focusing on implementation of the sensor module.

The used microcontroller will be presented in Section 4.1.1, followed by a study of the estimated cost in Section 4.1.2. After this, a brief exposition of some important notions is made as to better understand it's modules. The Inertial Measurement Unit is explained in Section 4.1.3 as is the Bluetooth BLE in Section 4.1.4. The characterization of the files that will be saved, and their transmission, will be clarified in Section 4.1.5 and 4.1.6, respectively. Finally, the code implementation will be stated in Section 4.1.7

4.1.1 Seeed Studio XIAO nRF52840 (Sense)

Diaz's project made use of a **Arduino Nano RP2040 Connect** microcontroller [Arduino, 2023], developed by *Arduino*. It was equipped with a Inertial Measurement Unit (**IMU**) containing an accelerometer and a gyroscope, both (both working in 3-axis), as well as a Wi-Fi module. In this section, the microcontroller that was used during the implementation of this project will be further discussed.

The new microcontroller used in this project has a smaller form-factor compared the its predecessor, which is beneficent considering the main goal of using it as a wearable device. Let's begin by going into a bit more detail about this device. As the first wireless product in the Seeed Studio XIAO family, the **Seeed Studio XIAO nRF52840** microcontroller [Seeed Studio, 2023] is equipped with a Nordic nRF52840 MCU which integrates Bluetooth 5.0 connectivity. It It also comes equipped 6-axis Inertial Measurement Unit (IMU). A micro SD card reading module has also been added to this device, in order to effectively guarantee a way to safeguard data. A detailed view of the boards and circuit scheme can be found in Appendix A. This scheme was made using the easyEDA tool [EasyEDA, 2023].

As Figure 4.1 shows, the device has been protected by a small box - this is the first step towards allowing the device to fulfill its goal of going underwater.

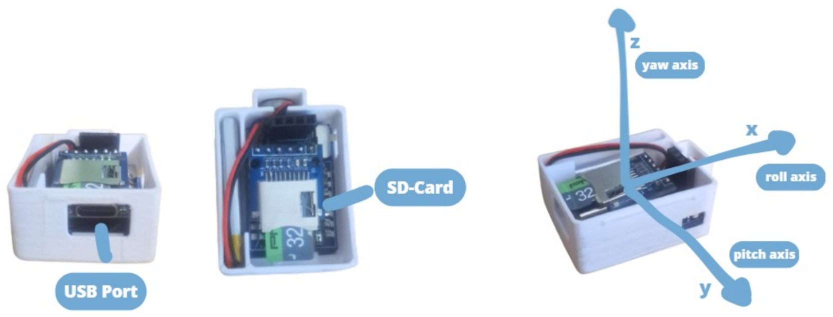


Figure 4.1: Sensor components and axis.

The SD card module sits on top of the Seeeduino microcontroller. This product comes to be about half the size of its predecessor. In future work, the

USB port will also be protected. The device is also shown with the three-dimensional axes system relative to the IMU device, preset by the sensor module itself. It'll will soon be made clearer why these axis are important to the problem in hands.

To sum it up, comparing the sensors of both projects, it's been possible to both keep the functionality guaranteed by the previous one, and also scale it down to a size that better fulfills the goals of the project.

4.1.2 Cost of the Project

One factor was missing from the project study in Section 2 - the cost. In order to determine the feasibility of this project, and to further compare to the previously mentioned related works, it is important to evaluate the manufacturing cost and compare it to the market price. The cost breakdown for the required components is shown in Table 4.1.

Table 4.1: Estimated project cost.

Component	**Cost**	**Reference**
Seeed XIAO nRF52840 (Sense)	€14.59	[Seeed, 2023]
Micro SD/TF memory card PNY 32GB	€7.99	[Worten, 2023]
MicroSD V1.0 adapter module	€0.49	[Aliexpress, 2023]
3.7V 450mAh lithium battery	€7.70	[Mauser, 2023]
PCB circuit board	€2.50	[jlcpcb, 2023]
Total	€33.27	

The total cost for the wrist sensor amounts to €33.27.

It is important to note that these prices are subject to change over time, and additional expenses such as transportation costs should be taken into consideration. However, when comparing the total cost to the available options in the market (refer back to Section 2), it becomes evident that this project offers a competitive advantage in terms of cost-effectiveness, making it a very viable option.

4.1.3 Inertial Measurement Unit - IMU

This chapter strives to better explain the “why”and the “how”regarding data extraction and interpretation. In order to obtain crucial information to the project, such as the number of strokes and to implement the recognition of swimming styles the first step is to actually *be* able to recognize strokes amidst the movement of the athlete. This is where the IMU module comes into play.

IMU stands for Inertial Measurement Unit. It is a device used in navigation and motion sensing applications to measure and report an object’s specific force, angular rate, and sometimes its orientation using a combination of accelerometers, gyroscopes, and magnetometers.

Accelerometers measure linear acceleration along multiple axes - as states before, this sensor works in a 3-axis basis - allowing the IMU to determine changes in velocity or movement in a specific direction. Gyroscopes measure the angular rate or rotation of an object, providing information about its orientation and angular velocity. Magnetometers detect the strength and direction of magnetic fields, which can be used to determine the object’s orientation relative to the Earth’s magnetic field.

By combining the data from these sensors, an IMU can provide information about an object’s position, velocity, acceleration, and orientation in three-dimensional space.

In regards to this project, it is used to calculate **yaw, pitch, and roll** angles, which represent the object’s orientation in three-dimensional space. Figure 4.2 provides a way to better visualize this rotations, specifically applied to the sensor

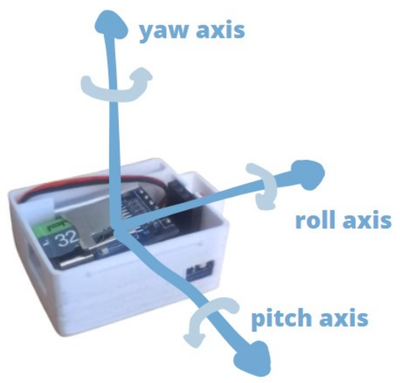


Figure 4.2: Yaw, pitch and roll as seen on the sensor itself.

In other words, roll, pitch and yaw angles refer the rotation around different axis - longitudinal, lateral and vertical, respectively. Typically measured in degrees.

The interpretation of these angles graphed along the time duration of the training session will allow the visualization of periodic movements that translate the athlete's strokes and swimming patterns, thus allowing the style identification by comparing the obtained graphs to those previously tested and classified - this will be further explain ahead.

To obtain accurate yaw, pitch, and roll angles, the IMU combines the measurements from the gyroscopes and accelerometers using sensor fusion algorithms.

In this project's case, it following the following line of thinking:

- By applying trigonometric functions, the acceleration on the three axes is calculated.

- The gyroscope reading are then integrated over time to obtain the object's current orientation angles. However, it's worth noting that gyroscopes are prone to drift, meaning they accumulate errors over time, leading to inaccurate long-term measurements.

- To compensate for said drift, a correction is introduced, factoring an 'average drift' calculated during calibration. A corrected value is obtained by subtracting the corresponding drift values from the gyroscope readings.

- Intermediate yaw, pitch and roll values that combine the corrected gyroscope readings with the accelerometer data are then obtained. These values are updated by adding the gyroscope readings minus the drift, divided by the time interval. This step helps reduce the effects of noise and drift, as the accelerometer data is reliable in the short term but prone to error in the long term, while the gyroscope data has the opposite behavior. The values are then filtered to improve accuracy and stability.

- Lastly, these intermediate are mapped to fit a [0, 360] range rather than [-180, 180].

All in all, this process takes into consideration the strengths and weaknesses of each sensor, aiming to provide the most accurate values possible.

Its worth noting, as mentioned, that the sensor has to be calibrated before starting the calculations, as its in this phase that the drift values are estimated.

With the code implemented during this project, the sensor is re-calibrated every time it's turned on - as it's part of the sensor's initialization process. Even though it only takes a couple milliseconds, it's still a section of code that doesn't need to be repeated every single time.

Due to this, a suggestion for future work would be to save the calibration results either in the SD card or on the Seeeduino's flash memory - the latter being a preferred option. The calibration could then be re-calculated periodically.

4.1.4 Bluetooth Low Energy Characteristics

The nRF52840 SoC on the sensor's board supports BLE - Bluetooth Low Energy and, it can operate in both central and peripheral modes (it can scan for and connect to other BLE devices as well as can advertise its presence and provide data and services to other devices). This section will further explain the importance of the component to the project.

Bluetooth Low Energy (BLE) is a wireless communication technology developed as an extension of the classic Bluetooth technology to cater to the growing need for energy-efficient connectivity.

Operating in a band that is available globally and that requires no licensing, it utilizes a frequency-hopping spread spectrum technique to mitigate interference from other devices operating in the same frequency band, which means it will be able to maintain a reliable connection even in crowded wireless environments.

Another one of the main characteristics of BLE is its low power consumption - as the name implies. Devices can broadcast data without establishing a full connection, it also implements various power-saving mechanisms such as reducing the duty cycle and optimizing data transmission to minimize energy consumption. All of this while also incorporating several security features to protect data and ensure secure communication - encryption and authentication mechanisms, including pairing methods like passkey entry, numeric comparison, or out-of-band pairing.

Most importantly, now relating directly to the project, it provides a standardized set of profiles and **services** that define the functionality of devices, let's focus on the profiles. These define collections of data attributes known as **characteristics**. Figure 4.3 better shows the structure.

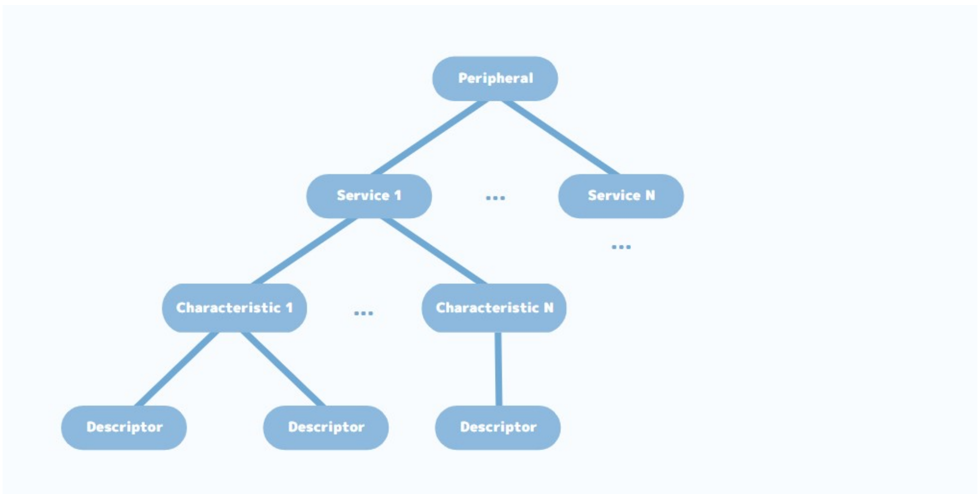


Figure 4.3: Bluetooth BLE hierarchy.

This hierarchical structure makes it easier to create interoperable devices and ensures compatibility between different manufacturers' products.

That is, by using pre-existing known characteristic names, we strive to keep the sensor compatible with other types of applications, in a way that the values received and transmitted are easily interpreted by both sides. Each profile, service and characteristic has a Universal Unique Identifier (UUID), a globally unique 128-bit (16-byte) number that can be used to track down these components.

The first tests regarding this functionality involved reading off the roll, pitch and yaw values via BLE on the 'nRF Connect' app, an application provided by Nordic Semiconductor to test capacities such as BLE. It was thanks to these initial tests, that the group gained a better grasp of how exactly services and characteristics related to one another, and the importance of using well-known names for both those components.

With the explanation of the sensor's state machine, in the next section, the implemented characteristics will be stated and justified.

4.1.5 Session Files

To further explain the need for a SD card we first need to establish exactly what the sensor will need to save.

As stated before, the rotation values in the three axes will allow the distinction between the four swimming styles - these values are the basis for all the calculations that concern this project, as can be used in a varied range of expansions in future work.

This process was the first step of development and involved an important decision - **should we store all the values in raw format?**, this decision would bring additional efforts when transferring data to the application due to its size. Should calculations be performed on the sensor itself? After all, sending only the results would allow for a easier transfer.

The option picked was to indeed store the data in raw format, for a very simple reason - scalability. Sending all the values, interval by interval, allows the creation of a system where data can be processed in various ways in the future, even those that are not foreseen in the current work.

Each session file follows the same two rules. Firstly, the name consists of the sessions starting time (in a HHMMSS format), and they're saved in a folder names after the session date (YYMMDD). This conforms to the fact that the SD cards that will be used do not allow names longer than 8 characters, while also supplying an extra level of organization. The second rule regards the content of the file. The values are stored in a CSV format - this allows for easier readings in a generalized format, and proved to be a big help when it came to debugging and validating values. The first line refers to the header standing as `timestamp;roll;pitch;yaw`, every following writings refer to the rotation values on the timestamp (in milliseconds) it specifies.

Listing 1 shows an example of the first few lines of a file, for better understanding.

```
timestamp;roll;pitch;yaw
5;250.10;141.63;49.75
14;249.46;152.78;49.34
24;247.45;162.21;48.36
33;246.12;172.17;48.32
42;244.87;182.12;48.23
...
```

Listing 1: Content of the session file

Thinking about the size of these files leads to the conclusion that a way to save large amounts of data is necessary. Storing the values in a separate SD card immediately seems to be a better option when compared to saving directly in the sensor's memory. The first method is suitable when one needs to store large amounts of data or log data over an extended period, and some sensors have limited internal memory, which means they can store only a certain amount of data before the need to retrieve it.

But to make an informed decision, let's find out exactly how much data will need to be saved. Let's think about the typical day of a professional or semi-professional swimmer. In a single day, up to three training sessions can be conducted, all of them can range from 1 to 2 hours, roughly. This brings us to the possibility of having a total of three 2 hour session files in a single day.

Table 4.2 states a estimated size for each value written and for the whole file.

Table 4.2: Bits allocation in a 2 hour file.

Value	Max. Number of Bytes
Timestamp	7
Roll	6
Pitch	6
Roll	6
Separators	4
Total	29

It's important to note that in C++ each character is equivalent to one byte.

Each new entry on the file starts with the timestamp value, meaning it's length will not be consistent throughout the entire file - at the start the values will be smaller. The maximum value saved would be 7200000 (2 hours in milliseconds) - this means 7 characters/bytes. The maximum rotation angle is 360$^{\underline{o}}$ - imagining the case angles stay consistently near this value, each angle would occupy 6 bytes as the value is saved as a floating point with two decimal cases - 5 for the digits and one for the decimal separator. Lastly, 4 extra bytes are needed to separate the values, in order to give them readability: three of those separate each value while another marks the separation of two samples. Each line can thus take up to 29 bytes.

Looking back on Listing 1, the first line (after the header) would occupy 21 bytes, while the second would occupy 22. So, in reality, these calculations result in an approximation since Every line can vary - but this also makes up for the size of the CSV header that's saved and that was not accounted for above.

The amount of lines in a file depends on the sampling frequency which, at the stage this study was conducted, had not been decided yet. The best way to truly test the limits of each solution is to find out the approximate file size with varying sampling frequencies. Diaz's project worked with a sampling frequency of 50Hz, this value served as a reference value for the study. Half and twice of that frequency were then also compared. Table 4.3 illustrates this analysis.

Table 4.3: Study in total file size.

Sampling Frequency	Total Size for 2 Hours
25 Hz	5 MB
50 Hz	10 MB
100 Hz	20 MB

To explain the calculations, multiplying the frequency by 29 bytes results in the amount of bytes that one second occupies. All that's left to do is to multiply it by 7200 - the amount of seconds in two hours - to have the approximate total file size for a session of this length.

Taking all this into account, let's now find out how much the sensor module can store. The official documentation states that it has "256Kb RAM, 1MB Flash, 2MB onboard Flash". As for the SD card, the storing capacity varies depending on the card chosen - the one given to the group for this project was a PNY 32GB Micro SD card [Worten, 2023].

We can thus conclude that the SD card is, as would be expected, the only feasible option. The microcontroller's memory can in turn be used to store simple variables like, for example, the BLE central name.

To finish this section, regarding the frequencies shown, it's true that maybe swimming doesn't require sampling frequencies as high as other sports, but that's exactly the point - one of the biggest goals set since the beginning was the creation on a solution that could be easily adapted to other sports. Sports like running, for example would require the choice of a high sampling frequency, hence the importance of implementing a system capable of supporting this type of data.

4.1.6 Data Transfer - USB or Bluetooth?

This section explains the study that was conducted to determine the best way to transfer the data from the sensor.

The transfer of files was a troublesome process that involved a lot of thought in order to pick the best option - there's a quite a varied range of choices all with their own strengths and weaknesses.

Two main potential options, Bluetooth and the Serial Port, were considered for data transfer. Bluetooth presented the advantage of being a user-friendly functionality without the need for cables or connections. However, an essential factor that needed to be taken into account was the transfer speed, especially considering the size of the files involved. To address this concern, a thorough study was conducted.

In order to evaluate the effect of file size on download speed, it was crucial to determine the maximum allowable file size, as explained in section 4.1.5. By establishing this parameter, we aimed to assess the potential impact it could have on the overall download rate.

Figure 4.4 illustrates the download time of various file sizes using Bluetooth and a serial port connection.

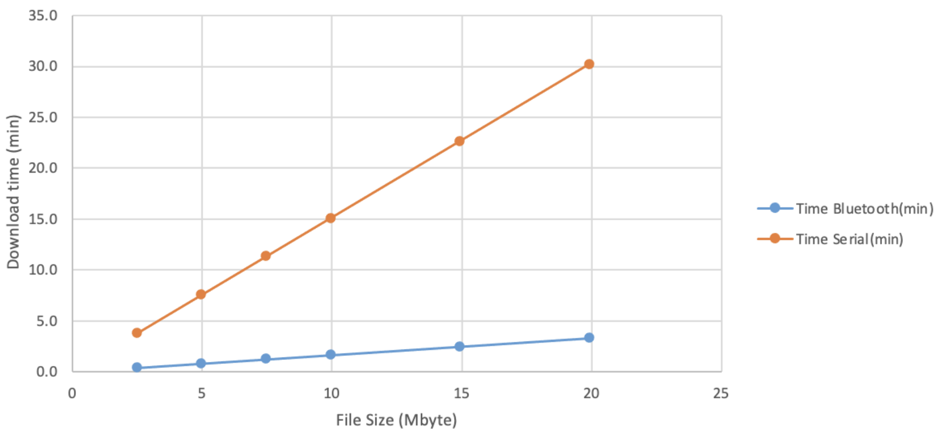


Figure 4.4: Sensor components and axis.

The estimated download time for a file can now be calculated via Bluetooth and USB serial port connections. The transmission speeds for each method can be found in their respective specifications. Bluetooth 5.0 offers an average transmission speed of 1 Mbps [NovelBit, 2023], while the serial port has a maximum transmission speed of 0.11 Mbps.

To calculate the file download speed, we convert the file size from bytes to bits and multiply by the transmission speed.

Based on the aforementioned considerations, it would be concluded that data transfer via Bluetooth was the preferred option. The convenience of use, combined with its superior speed, which is 10 times faster compared to the serial port, leaves no room for uncertainty.

However, during the implementation of Bluetooth communication, it became apparent that more than just establishing a connection was required. It required a protocol to facilitate data transmission - and this was no simple task as it had to conform to existing sets of rules. This prompted the group to begin a search for libraries that would enable this type of data transfer.

During thorough research, no libraries were found that met the specific criteria. Developing a custom communication protocol from scratch would require significant time and effort, making it unfeasible within the available resources. Consequently, the decision was made to abandon the usage of Bluetooth, despite its recognized superiority as the optimal choice and should be pursued for future work. As a result, the focus was redirected towards data transfer via the USB serial port.

In order to enable data transfer via the USB serial port on iOS, we encountered a roadblock as it required a special permission from Apple. Unfortunately, this restriction prevented the group from displaying the data as intended on both Android and iOS platforms. To overcome this challenge, and after speaking to the advisors as this majorly affected the original course of the project, a solution was divised: incorporating a web platform into the equation. This approach ensured that we could maintain compatibility across both iOS and Android, allowing the group to access and view training information on the web. By connecting the microcontroller to a computer at the end of each training day, all the training data could be effortlessly downloaded. In the following sections, the logic of data transfer will be presented in detail for both the sensor side and the web application side, respectively in Sections 4.1.7 and 4.3.

4.1.7 Sensor State-Machine

With the sensor module components and some crucial notions explained, this chapter will introduce the code implemented **state-machine** that controls the sensor module's behavior. The module currently functions according to the state-machine are displayed in Figure 4.5.

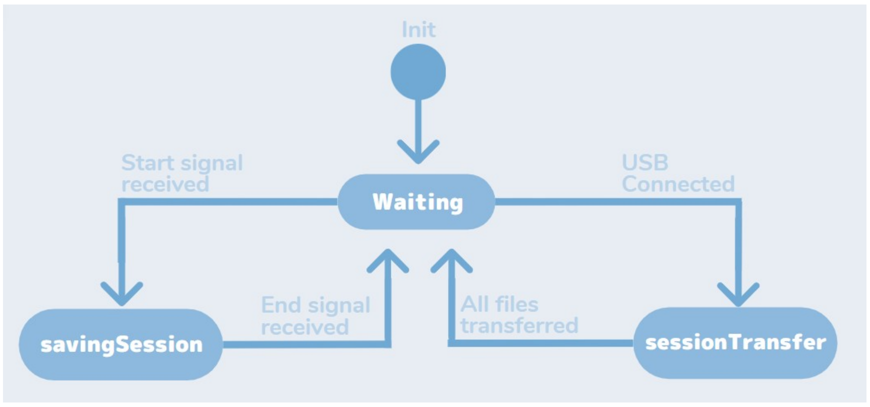


Figure 4.5: Sensor state-machine.

Let's further explain each of these states.

init This state stands as a way to prepare every module and component that will be used in the following states. This step can be divided into phases. The first phase takes care of sensor-specific initializations, this refers to LEDs and other debugging facilities, and also to the IMU configuration and calibration.

Following this phase, the SD card module must be prepared. This project makes use of the built-in SD library from Arduino, which enables reading and writing on SD cards. It is compatible with all architectures making it viable for any future changes. During this stage, several verifications must be completed to ensure the sensor's functionality: it begins by setting up the connection to the SD card, by supplying the pin it's attached to (for this project, pin 3 was used), it then verifies if the directory where sessions will be saved exists and is not corrupted - the directory is created if it does not, in fact, exist. Any errors that arise from this phase block the usage of this component until something is done to fix it and the sensor is reinitialized. The user is informed of what happened via a LED blink and a displayed message.

Finally, all that's left is the initialization of the BLE module. This means setting up the BLE central's name (**all devices intended to be used for SWIMU must include that 'SWIMU' in the name**, this is the name that other devices will see when looking for centrals to connect to, and the mobile app will filter the names as such). It proceeds to prepare all the services, characteristics and onWrite functions that will be necessary throughout the operation. It finished off by advertising the central and waiting for connections - this is when the sensor enters the `waiting` state.

waiting The `waiting` state is the first state the sensor falls into in the state-machine - all it does is verify if the BLE central is connected to another device, processing all events and actions that concern this module, and check if any data has been sent via Serial port.

savingSession Let's begin with the transition to the `savingSession` phase. As will be seen in the section regarding the mobile application (Section 4.2), the start of a session is marked with the sending of the timestamp such request was made, it will also send the pool size in meters. The sensor thus needs to ensure the advertising of a characteristic that allows for these type of values to be received: The **Current Time Service (CTS)** is a standard service defined in the BLE Generic Attribute Profile (GATT). It provides a way to retrieve and update the current time on a BLE device. There's no distance related characteristic by default, but a custom one can easily by created: as stated before each Service and Characteristic has a specific UUID, custom **Distance Service** and **Distance Characteristic** are defined using a unique indentifier. The creation of a function that gets called once values are written on the time characteristic serve as a way to immediately attempt to create a file with the correct nomenclature and the header line (refer back to Chapter 4.1.5). If the creation is a success, the sensor enters the `savingSession` state. During this stage the sensor will continuously read the roll, pitch and yaw values from the IMU and print them in the session file (as stated in Section 4.1.5).

To optimize efficiency, the file is saved as a global variable, reducing the need for repetitive opening and closing. This approach allows for a sampling frequency of about 100Hz - this is roughly twice the frequency used in the previous project. This frequency seemed to get the most accurate results in stroke calculation and style identification. One danger that arises from keeping the file always open is the fact that any problems that happen to the sensor amidst the session will promptly destroy all data that had been saved. To mitigate this, every minute the file will be flushed. Flushing a file refers to the process of writing any buffered data from memory to a specified file. When data is written to a file, it is often stored in a buffer in memory for efficiency purposes. This call allows the system to physically write the buffered data to the file.

The file is only closed when the central receives a signal that indicates the end of a session - also given by the sending of a timestamp. It promptly returns to the `waiting` state.

sessionTransfer Regarding the `sessionTransfer` state, to which the sensor defaults to once serial details received, it follows a basic communication protocol as shown in Figure 4.6.

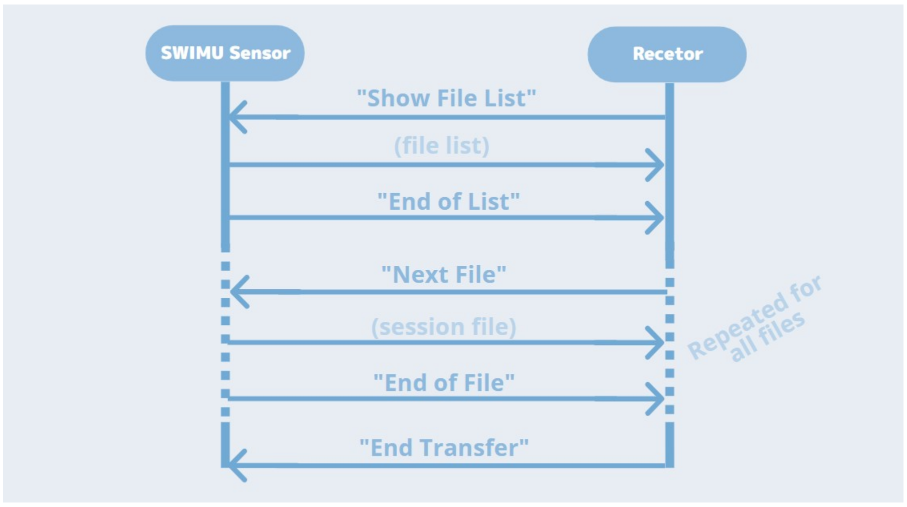


Figure 4.6: Communication protocol.

Very simply put, it waits for a signal that triggers the beginning of the transaction, and responds back with a list of available files (files that have yet to be transferred). The receptor then proceeds to ask for the files one by one.

To facilitate the transfer, a buffer array is created to be used as a means of storing chunks of data read from the file.

Inside a loop, chunks of data are read from the file and transmitted as bytes via USB. It ensures that consecutive newlines or carriage returns are not treated as separate lines in the serial output, and does not “break”a line in between two separates chunks. The loop continues until the entire file is read.

Once the loop ends, the file is closed and then deleted - if its deletion leaves an empty date directly, that directory is also deleted.

It finishes up by sending a message to the serial connection to indicate that the transfer is complete.

It repeats this until all requested files are transferred, then returning to the waiting state.

This protocol is then independent of the type of device that the device communicates with, as long as the message structure if followed.

4.2 Mobile Application

This section describes the implementation process for the mobile application. As stated in Section 3.4, where the project architecture was shown, it was made clear that the main goal of the mobile application was the communication with the sensor via Bluetooth BLE.

First, the the set up process will be explained in Section 4.2.1, followed by the implementation of the application itself in Section 4.2.2.

4.2.1 Setting up React Native

In order facilitate greater usability for a broader user base, it was determined from the start that implementing both iOS and Android platforms for the project was necessary.

To streamline the development process and avoid the need for separate coding for each platform, it was decided that utilizing React Native [React Native, 2021] would then be the optimal solution for the project. React Native is used to build applications that run on both iOS and Android devices (that is, cross-platform mobile applications). Initially, the Expo Go Quickstart was utilized, this incorporated React Native and provided an easier learning curve for beginners. However, a limitation was soon encountered, since Expo was incompatible with BLE Manager libraries which were necessary for this module. To solve this limitation, the transition to the React

Native CLI Quickstart was made, which proved to be a suitable alternative and functioned as intended for the project.

4.2.2 Implementation

The mobile application as implemented in this project consists of two pages, each serving a different function. Let's delve into each of them. Figure 4.7 shows the progress from the mock ups to the final product.

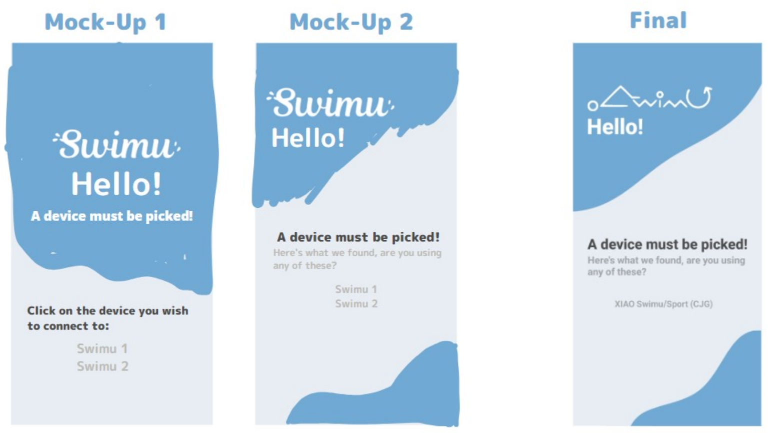


Figure 4.7: First page of the mobile application.

A SVG library was used to draw the blobs seen in the figure above. Regarding the logo used, to explain the change, the mock ups used a logo created by the group this year as a fill-in - the final version uses the original logo created for the original SWIMU project. The purpose of this page is to look for BLE centrals to connect to, and allow the user to pick the one that will be used.

Let's break down the implementation. The `react-native-ble-plx` library [Github, 2021] was used - this is a third-party library specifically designed for interacting with BLE devices within a React Native app. It provides a set of APIs and components that enable developers to perform tasks such as scanning for nearby BLE devices, establishing connections, reading and writing data to device characteristics, and managing Bluetooth services. The library abstracts the underlying platform-specific BLE APIs and provides a easy-to-use interface for working with BLE devices.

Once launched, should Bluetooth related permissions not have been given, they must be first requested to the user. It checks for Bluetooth scanning permission, Bluetooth connection permission, and access to fine location permission. The Bluetooth scanning permission allows the app to search for nearby Bluetooth devices. This is crucial to permit pairing. Secondly, the Bluetooth connection permission enables the app to effectively establish connections with Bluetooth devices. Lastly, the access to fine location permission grants the app access to precise location information. While not directly related to Bluetooth, it may be needed for location-based Bluetooth services or features that rely on accurate geographical coordinates - this allows for a more generic implementation. If these permissions are granted, the scanning process for Bluetooth devices is started. It specifically looks for devices whose name includes the name 'SWIMU', as to not show the user "useless"devices (or devices that can't be used). This process works in the background.

A list of all available devices is presented to the user, updated whenever a new device is found, not without first checking if any duplicate devices were listed (a small mistake that often happens while scanning).

Each item in the list functions as a button - once clicked, it means the user chose that device as the one they want to use. And so, a connection is established with that specific Bluetooth device. If the connection is successful, the connected device is stored in the in a state variable and the device scanning process is stopped. After that, the function checks the connected device in order to discover all available services and characteristics - in this implementation, this means finding the characteristics that were advertised on the sensor module, such as the time characteristic. Finally, should the needed characteristics be found and no errors occur, the user will be redirected to the next page.

This new page, and its planning process, is seen in Figure 4.8.

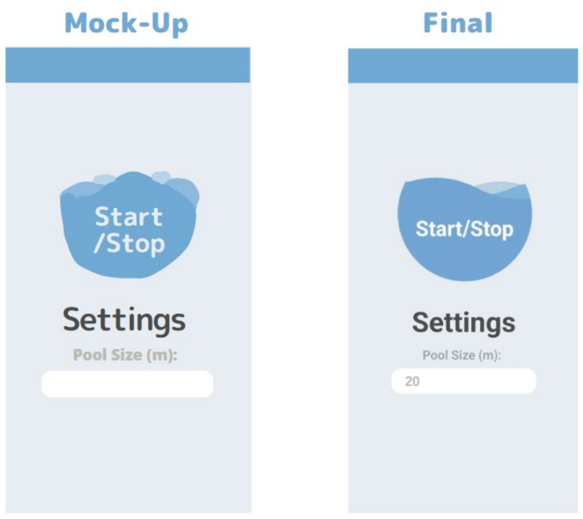


Figure 4.8: Second page of the mobile application.

The start/stop button is an animated SVG using the `react-native-wave` library [Github, 2018] as to simulate water movement.

The functionality of this page is very straight forward: clicking on the button sends the current timestamp to the connected device via the BLE **Current Time Characteristic**, toggling the data saving state (either begins or ends a session). Moreover, the user can send the pool size via the custom **'Distance Characteristic'**, as stated in Section 4.1.7.

In future work, this page can show a button that opens a popup that allows users to state their goals for the swimming session, allowing for the planification of the distribuition of swimming styles throughout the session, for example.

4.3 Web Application

The web application encompasses a range of crucial functionalities, including data retrieval, data processing, and data visualization. It acts as an interface between the user and the stored training data, providing a user-friendly platform to access and analyze the collected information.

This section will be delving into a comprehensive overview of the web application in Section 4.3.1, before better exploring its various components in Sections 4.3.2 and 4.3.5. Section 4.3.4 elucidates the processes involved in data collection from the cloud service.

Finally, Section 4.3.6 will then proceed to explain how the data is presented to the user visually, while Section 4.3.7 and 4.3.8 explain the analysis and calculations necessary before this display can be made.

Furthermore, the challenges and limitations encountered throughout the implementation phase will be addressed.

4.3.1 Structure and Communication

The web application is divided into two main components: the client and the server. It thus follows a client-server architecture. This section will delineate the function of each one.

The files containing the behaviour of each component are separated in two folders (server and client) to further increase comprehension and modularity.

The client component refers to the user interface or front-end of the web application, rendered by the web browser. It is responsible for presenting the application's visual layout and interacting with the user. The client component runs on the user's device (such as a computer or mobile device) and is responsible for sending requests to the server and receiving responses. See Section 4.3.6 for more details regarding the implementation.

The server, on the other hand, refers to the back-end of the web application. It handles the processing and requests for storage of data, as well as the logic and business rules of the application. The server component typically consists of a web server, an application server, and a database. It receives requests from the client, performs the necessary operations, retrieves or updates data from the storage component, and sends the response back to the client.

The communication between the client and the server takes place over the internet using HTTP, a standard communication protocol. HTTP operates on a stateless request-response cycle, where each interaction is independent. Requests include information such as the HTTP method, headers, and URI to specify the desired operation and resource. Servers process the requests and generate appropriate responses, which are returned to the clients. HTTP defines methods like `GET`, `POST`, `PUT`, and `DELETE` for different types of operations.

A URI is a string of characters that identifies a particular resource. Endpoints are part of the URI structure and typically follow a specific pattern that maps to the desired resource or action. For example, in project the endpoints are defined as `/SwimuWeb/SessionList`, `/SwimuWeb/ShowSession`, and `/SwimuWeb/DeleteFile`. Just reading each of these, its easy to gain an idea of what each one does. These are the URIs that the clients can use to access the corresponding resources or perform specific operations on the server.

In Section 4.3.2 a better explanation of how these endpoints are implemented will be given.

Now to explain the flow, the client sends requests to the server, specifying the action it wants to perform (for example, retrieving a training session or submitting a new one), and the server processes the request and sends the appropriate response back to the client. Section 4.3.2 will help gain a more detailed understanding of this component.

Figure 4.9 shows a overview of the endpoints and the expected response.

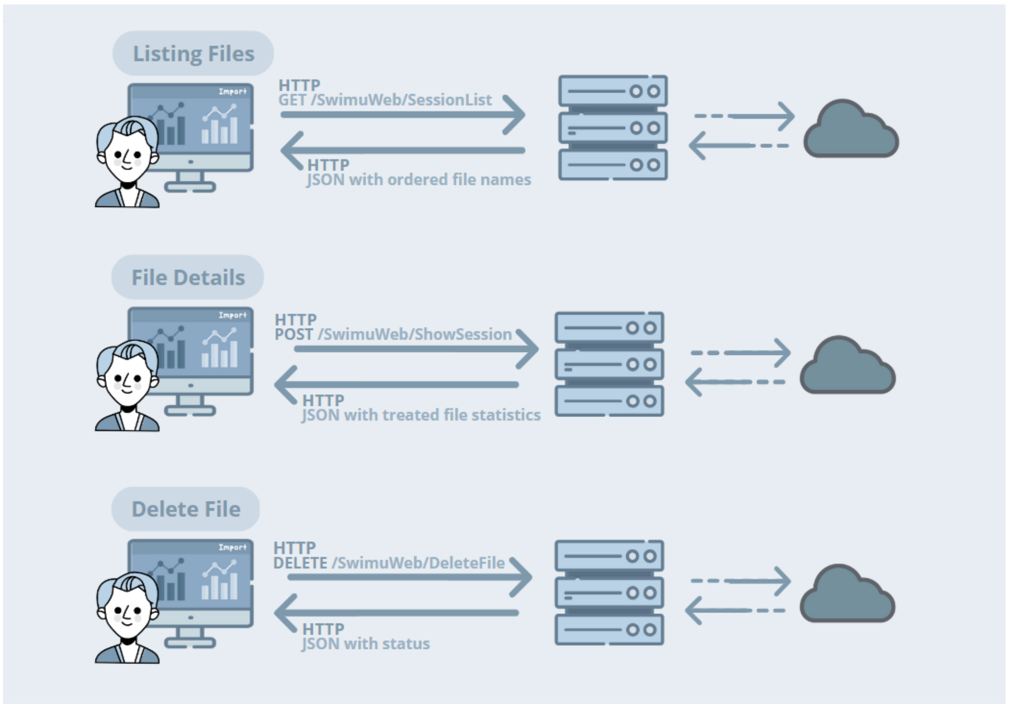

Listing Files
HTTP
GET /SwimuWeb/SessionList
HTTP
JSON with ordered file names
File Details
HTTP
POST /SwimuWeb/ShowSession
HTTP
JSON with treated file statistics
Delete File
HTTP
DELETE /SwimuWeb/DeleteFile
HTTP
JSON with status

Figure 4.9: Client-server communication via HTTP.

The communication between the server and the Cloud service will be elucidated in Section 4.3.4, it's accessed by the server in order to complete the client's requests, before sending its response.

Overall, the client-server model forms the foundation of the web application architecture, enabling the separation of concerns between the user interface and the back-end functionality, and facilitating the exchange of data and communication between the two components. In Appendix B, a visual reprensentation of the application modules and functions can be found.

4.3.2 Server Setup

The server acts as an intermediary between the browser and stored files, it's the back-end of the application. It's also in charge of requesting and downloading files from the sensor module once the user establishes a USB connection. In this section, the set up and implementation will be delineated. Let's start by explaining how the server is set up and deployed, and how it permits communication with the client.

The server is set up using Node.js [Node.js, 2023]. This is a open-source, cross-platform JavaScript runtime environment that allows developers to run JavaScript code outside of a web browser, making it ideal for server-side applications. Node.js uses an event-driven, non-blocking I/O model, which means it can handle concurrent requests efficiently without blocking the execution of other tasks. This design makes Node.js highly scalable and well-suited for building real-time applications - all of these characteristics fit the goals of the project. Another key advantage of Node.js is its ability to use JavaScript on both the client-side and server-side, enabling developers to write full-stack applications using a unified language. Furthermore, it provides a set of built-in modules and packages through its package manager, `npm`, which allows to easily extend the functionality of the applications by leveraging the vast ecosystem of open-source libraries. Using this framework, a server was created that utilizes an event-driven architecture, where callbacks or promises are used to handle asynchronous operations. This asynchronous nature allows the server to handle a large number of concurrent connections efficiently, building the foundations of a scalable and well performant applications.

Regarding the implementation of the routing mechanism, the Express framework is used [Express.js, 2023]. Express is a web application framework for Node.js. It provides a set of features and tools that simplify the process of building web servers and APIs. Express is designed to be unopinionated, allowing developers the freedom to structure their applications according to their preferences. At its core, Express provides a routing mechanism that allows developers to define routes and handle HTTP requests for different endpoints. It supports various HTTP methods like `GET`, `POST`, `PUT`, and `DELETE`, etc., enabling the creation of RESTful APIs. Express also provides middleware, which are functions that are executed in the request-response cycle and can perform various tasks such as parsing request bodies, for example.

Upon import the necessary modules, the server is created by invoking the `express()` function, which returns an instance of the Express application. The body-parser module is utilized to parse incoming request bodies in JSON format. The code then specifies port on which the server will listen for incoming requests as port 3000. It then creates an HTTP server using the `createServer` function from the built-in HTTP module, passing the Express application (server) as an argument. Listing 2 shows this behaviour.

```
// Imports

const srv = express();
const jParser = bodyParser.json();
const port = 3000;
const http = createsrv(srv);
serialStart();

srv.get("/SwimuWeb/SessionList", jParser, async (req, res)=>{
  try {
    const fileList = await listsrvFiles();
    res.setHeader("Content-Type", "application/json");
    res.send(JSON.stringify(fileList.reverse()));
  } catch (err) {
    console.log("Error", err);
    res.status(500).send("Error retrieving file list.");
  }
});

// Set the rest of the endpoints...
```

Listing 2: Server setup example

As shown above, the aforementioned endpoints and routes are defined using methods provided by Express.

- The first route, `GET /SwimuWeb/SessionList`, is responsible for retrieving the list of file names saved in the Cloud storage, it's ordered from the newest to the oldest file. The obtained list is then sent as a JSON response to the client.

- The second route, `POST /SwimuWeb/ShowSession`, is used to retrieve data related to a specific file, in order to later present it on the server side. When a request is made to this endpoint the requested filename is passed in the body. It is used to retrieve that session's statistics (for example, the number of strokes, laps, SWOLF score, and more). A JSON response is sent to the client.

- The third route, `DELETE /SwimuWeb/DeleteFile`, is responsible for deleting a file from the server (for example, necessary if a session sufferend any anomalies). A success message is sent as a JSON response to the client upon successful deletion.

When a request is made to any of the endpoint, the code asynchronously calls a function that interacts with an Cloud service in order to fulfill the purpose of the request (see Section 4.3.4).

Finally, the server starts listening on the specified port by invoking a `listen()` method. After this, the server will be continuously listening for incoming connections.

4.3.3 File Transfer

In Section 4.1.7, the Serial communication protocol was defined. In this section, the explanation by presenting the receiver side will be concluded. These operations are separated in a module called `serial-utils.js` as Figure 4.10 shows:

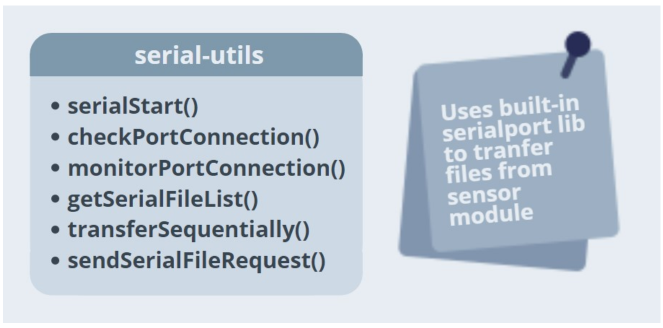

serial-utils
• serialStart()
• checkPortConnection()
• monitorPortConnection()
• getSerialFileList()
• transferSequentially()
• sendSerialFileRequest()
Uses built-in serialport lib to tranfer files from sensor module

Figure 4.10: Javascript module in charge of serial utilities.

It starts by checking the connection a specified serial port at regular intervals (first three metods). Once the port is connected, it attempts to begin the protocol by requesting the list of files available (`getSerialFileList()`). Upon retrieving the list it proceeds to transfer each file sequentially, via Serial communication, to the server (`transferSequencially()`). A request is sent for each file (`sendSerialFileRequest()`), ensuring that it's transferred one by one and in its entirety. If an error occurs the process is stopped and the communication is closed. Files will not be lost "midway"as they're only deleted from the sensor module upon a successful reception.

Once the content is received, two versions of it are stored in the Cloud service: One is saved in it's raw form - this allows for the session information to remain untouched and available for future operations that had not been foreseen in the current project, giving room for expansions. Another version refers to a JSON containing the results of the data treatment process (as explained in Sections 4.3.7 and 4.3.8). This helps reduce the time necessary to retrieve and avoids the need for redundant processing every time a user wants to view a session. It also makes it easier to compare between different sessions.

Lastly, it's important to note that this process is asynchronous, functioning in the background without blocking the user's access to the rest of the web pages' functionalities.

4.3.4 Cloud Service

This section will now explain how the files will be accessible from withing the web application. Related operations are separated in a module called `aws-utils.js` as Figure 4.11 shows:

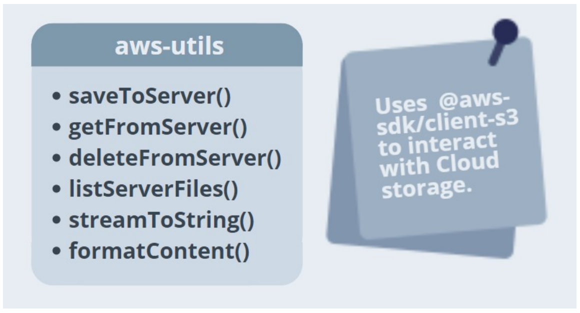


Figure 4.11: Javascript module in charge of Cloud service utilities.

To ensure continuous availability of both the raw and tretaed data for the web application, the group opted for the utilization of the free AmazonS3 web service provided by Amazon [Amazon, 2023]. There is a very wide range of options that can be found regarding services that provide these kind of remote storage capabilities. But truth be told, apart from the price most of these options barely differ from one another. AmazonS3 is, however, commonly seen as the most popular one.

The setup is rather simple: all that needs to be done is the creation of a Amazon Cloud account. Upon this creation, a key will be associated to the account - this is the key that will be used for any write, read or update requests sent to the service as a means of authentication.

This service allows a way to securely store the files acquired from the sensor module in dedicated buckets created specifically for the SWIMU project. In the context of cloud storage, a bucket refers to a logical container or storage space where you can store and organize your files or objects, much like a folder in a typical file manager application. Two buckets are created, as stated before, two file versions are saved: one containing the raw data, and another

By utilizing this approach, all the files are reliably stored and easily accessible whenever needed.

As can be assumed from everything that's been explained thus far, this project did not undertake the task of allowing user accounts. Now's the time to quickly address this.

If in a future extension of this project this feature is implemented, this is the only aspect of the current solution that needs to be adapted - more buckets must be created, two for each user. The database that holds the user's information (for example, one could use Firebase) can also hold a reference for that user's bucket names.

The following functionalities have been implemented regarding the access and manipulation of the files from the Cloud service:

Saving Files This method saves a file to the S3 server. It creates a parameter object containing the file name and the bucket it stands in and utilizes the s3Client to send a `PutObjectCommand`, which saves the object to the server. This method is called twice from the SWIMU application whenever a new file is downloaded from the sensor module;

Obtaining Files This method retrieves the content of a file from the S3 server. It creates a parameter object with the wanted file name and the bucket it stands in and utilizes the s3Client to send a `GetObjectCommand`, which fetches said object from the server. It's called whenever the information of a file needs to be displayed;

Deleting Files This method deletes a file from the S3 server. It creates a parameter object with the name of the file to delete and the bucket it stands in, and utilizes the s3Client to send a `DeleteObjectCommand`, which removes the object from the server;

List Files This method lists the existing files on a S3 server bucket. It creates a parameter object with the bucket name and utilizes the s3Client to send a `ListObjectsV2Command`, which retrieves a list of objects from the server. The method then returns the list of files.

By implementing these methods, we can effectively interact with the AmazonS3 web service, ensuring seamless storage and retrieval of the sensor-generated files for further processing and analysis. With them being all relatively generic, the same method can be used for multiple functionalities in the present and future work.

4.3.5 Client

A module is created on the server to include functions for fetching data from the specified endpoints, allowing the display of the fetched data on the web page, the functions are called as the user interacts with the buttons on the page. This section will briefly explain how the requests are prepared and sent.

For each endpoint previously mentioned, a function is created. The file list request is sent as soon as the user loads the page, allowing the near-immediate display of the list so that user's can navigate their sessions. File detail requests occur in two ways: the first one is automatically executed to show the most recent file, as soon as the list request is fulfilled, the second way is prompted when users choose a new file to view. Finally, the delete request is also only prompted once the user presses a delete button. More on this can be found in Section 4.3.6, where the visual implementation is described.

Regarding the process of sending requests, the Fetch API is used. This is a modern web standard for making network requests in JavaScript. It provides a more flexible alternative to the older `XMLHttpRequest` object. It's built around the concept of Promises, which allows for more concise asynchronous code.

The previously mentioned functions all follow the same process:

1. Two variables are defined: one stating the endpoint and another containing the request options. In other words, the first one holds the URL to which the request will be made while the options variable is an object containing configuration options for the request. This step allows the specification of the request method and sets the `Accept` and `Content-Type` headers to indicate what the client expects to send and to receive (for example, JSON content);

2. The function `retryFetch` is called, this is a custom implementation that performs a fetch request with retry logic. It allows for retries in

case of network failures or timeouts. Here's a breakdown of how the function works:

- The maximum number of retry attempts and the delay between them is specified;
- The function returns a new Promise that encapsulates the entire retry logic, inside of which the request is attempted and the response is processed;
- Any received response is checked to see if it was successful (status code in the range 200-299). If it is, planned retries are stopped and the response data is extracted. If the response is not successful, an error is thrown with the status text of the response;
- If any error occurs during the fetch process and there are remaining retries (and a response has not been received yet), a `setTimeout` function is used to schedule another fetch call after the specified delay. The number of retries is decremented, indicating an attempt has been made;
- If all retries have been exhausted without a successful response, an error is logged, and the Promise is rejected with the captured error;

3. The code uses the aforementioned Promise's `then` method to handle any successful responses;
4. The received data is then treated as necessary, in order to show the user the results;

Overall, this module uses the Fetch API to make an HTTP requests, handle the response data, and handle any potential errors that may occur during the request.

4.3.6 Data Presentation

In this section it will be made clear exactly **what** the web application will show - after all, the main goal for this project is to allow the user to view their progress in a more user-friendly and aesthetic way, via a dashboard.

The final product contains the two main dashboard pages, as visible is Figure 4.12 and 4.13. **All values shown were part of a test using Mock Data**, and may not translate to accurate values as would be seen in a real session:

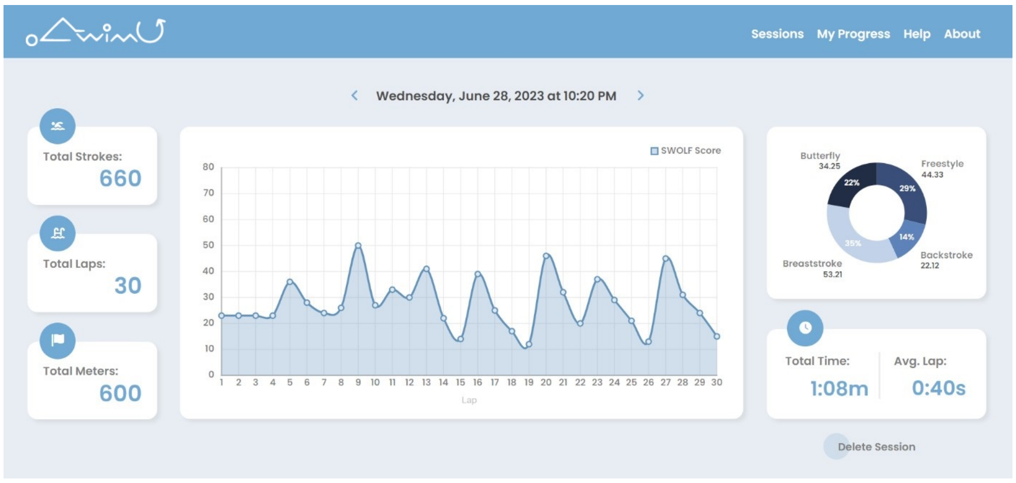


Figure 4.12: Swimming journey page.

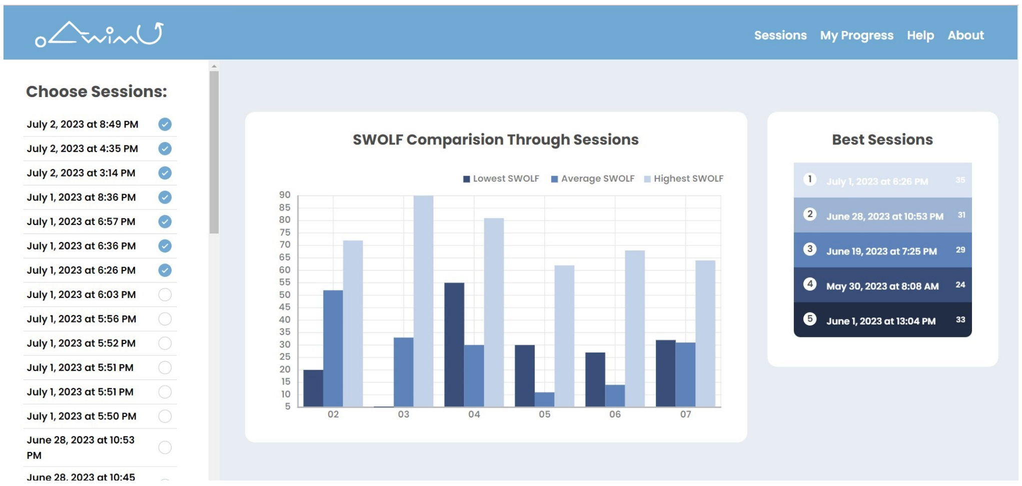


Figure 4.13: Session page.

Let’s detail all the functionalities of each page.

The first page works as a 'My Swimming Journey' page, meaning users will be able to see the details of all of their sessions, ordered. The following actions are possible:

1. **Choice of File**: The user will be able to iterate through the files saved on the Cloud Service and pick which one will be displayed. This can be done via navigation using the arrows (which leads the user to the previous and next file, respectively), or by clicking on the displayed file name to show a dropdown menu with all available files. The shown statistics will be automatically updated, without the need to refresh the entire page;

2. **Total Number of Strokes**: As calculated using the method described in Section 4.3.7;

3. **Total Number of Laps**: As calculated using the method described in Section 4.3.7;

4. **Total Distance Swum**: As estimated by multiplying the number of laps by the size of the pool, in meters;

5. **SWOLF by Lap**: The graph in the middle displays information regarding the SWOLF score obtained in each lap, as calculated in Section 4.3.7. When hovering over the graph, it also displays the style that was identified in that lap (Figure 4.14), obtained as explained in Section 4.3.8.

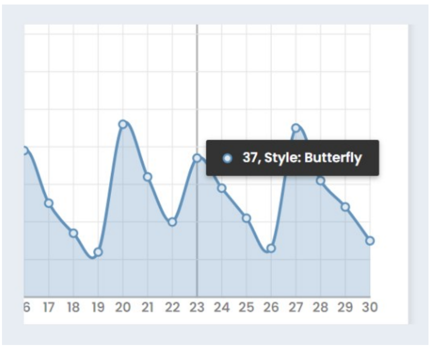


Figure 4.14: Hovering over a value on the SWOLF graph.

As different dataset sizes call for different graphing styles, this graph will be a Bar Chart when there's up to 12 laps (Figure 4.15), after that, it will be displayed in a Spline chart (as seen before in Figure 4.12);

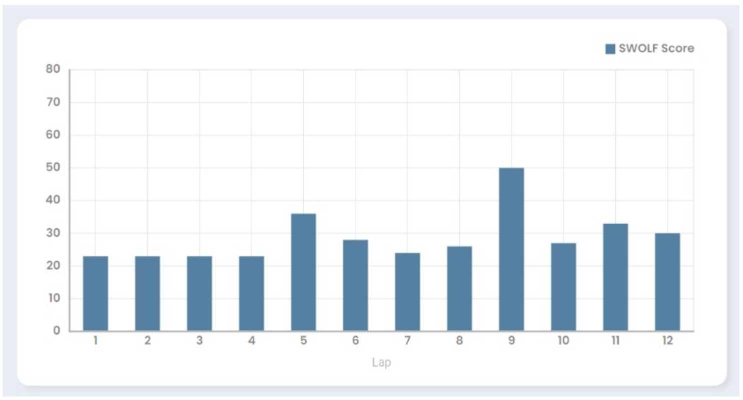


Figure 4.15: Bar chart variant.

6. **Swimming Style Distribution**: Donut chart displaying the distribution of swimming styles trained throughout the session, calculated as explained in Section 4.3.8;

7. **Time Statistics**: Total training session time and estimated average time per lap;

8. **Delete Session**: Delete current training session from the Cloud Service, useful when a session was done by accident or when any anomaly happens;

Regarding the second page, (refer back to Figure 4.13) it works as a means of comparing different training sessions by SWOLF score. Let's review: the SWOLF score is independent of the specific swimming style being used. It applies to any stroke, including freestyle, backstroke, breaststroke, and butterfly (although it's more commonly used for freestyle). The score reflects the swimmer's ability to minimize the number of strokes taken and the time taken to complete the length, regardless of the stroke technique employed. This means its a value fit for session comparison.

As such, this page provides a way to view the progress throughout multiple sessions side by side.

On the tab to the left the user is able to pick from a scrollable list which sessions they want to view on the graph (only up to 7 sessions can be displayed). The first five sessions are selected immediately as the user enters the page.

The graph itself displays the highest, lowest and average SWOLF score obtained during that session, allowing the user to toggle between which of these values they'd like to display.

Lastly, on the right side, the page displays a ranking tab best selected sessions in regards to average SWOLF score obtained - The best session is the one with the lowest SWOLF score and where the highest and lowest SWOLF score obtained differ the least from the average, signifying a consistent efficiency.

Now's the time to better explain how these charts are created and filled.

The graphs were created using the DHTMLX [DHTMLX, 2023] Charting library. This library permits the creation of a wide range of charts. Also allowing the use of various sub-types, synchronous representation of several data properties on the same chart, both 2D and 3D presentation, different data sources, interactive actions and tooltips. It is completely written in JavaScript and ready to be used in any HTML page. The only step that needs to be fulfilled is the download of the licensed library - this download, as used in this project, provides a one month free trial.

The documentation provides all the information needed to format and style the graphs to this project's needs. After setting up the graph, all that's left is to add the values as necessary - these are obtained from the JSON versions from the Cloud Service, since these values have already been treated and cut a bit of processing time.

4.3.7 Data Treatment and Calculations

This section aims to explain the process necessary to calculate the number of strokes and laps in a swimming session and, ultimately, the SWOLF score obtained in each lap. The entire process is separated in a module called `data-utils.js` as Figure 4.16 shows.

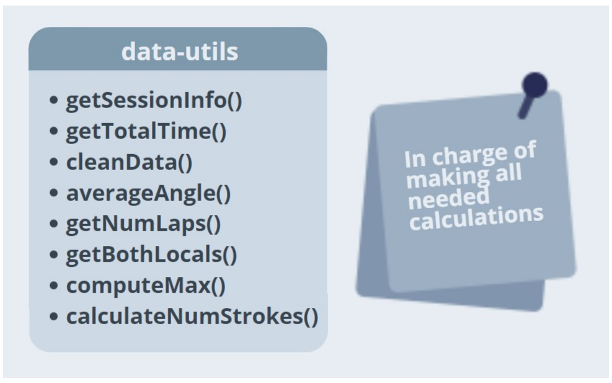


Figure 4.16: Javascript module in charge of data treatment utilities.

The various operations are separated in different methods, with illustrative names to facilitate comprehension. Let's delineate why these methods are necessary.

First, it's important to take a look at the data received from the sensor module, focusing, for example, on the roll results, as seen in Figure 4.17. This data was obtained from a small simulation of a training session, without going into the pool:

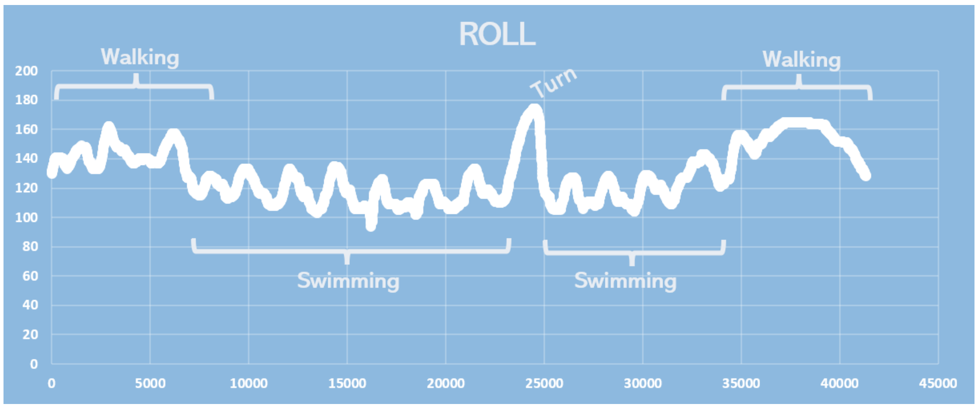


Figure 4.17: Roll signal obtained for a swimming session.

This training session followed as such: it started by walking from the phone used to start the session to the place where the strokes were simulated. About midway through the strokes the end of a lap was simulated by recoiling the arm next to the body, to get a similar effect to that of a turn. It then ended with a simulation of walking, as well. From the graphics above, all these moments are clearly distinguishable from each other and from the swimming periods, where the movement is clearly periodical. The same can be said for the pitch and yaw signals.

As such, there's some parts of the data that must be removed in order to count the stroke and lap amount. This corresponds to the duration of time in which the swimmer hasn't started or has just completed the swimming cycle, but the device is still collecting sensory data from other types of movement (in other words, the user is walking). This situation then correspond to a series of unknown and variable initial and final seconds of the swimming cycle.

In order to achieve the removal of these situations, the detection of local maxima or minima in the obtained angles will be performed.

Here's a brief explanation of all the steps performed:

1. **Average Signal**: Calculation of the average angle for each time instant in order to obtain an average signal. This means, for each timestamp, adding the values obtained from roll, pitch and yaw rotations dividing the result by 3. The signal obtained will still be periodical during the swimming cycle;

2. **Local Maxima**: The local maxima of the average signal will be calculated. Local maxima must have at least half the amplitude of the difference mid-point between the global maximum and the global minimum of the average signal. Furthermore, maximas can only be considered should there be at least 100 instances (equivalent to 1 second considering the frequency of 10ms mentioned earlier) between two consecutive maxima. This number of instances is an arbitrary value but is considered feasible as a minimum interval between two consecutive strokes performed by the same arm (Listing 3);

```javascript
function computeMax(data, clean = false) {
  let locals = [];
  const gMax = Math.max(...data.map(obj => obj.avgA));
  const gMin = Math.min(...data.map(obj => obj.avgA));
  const limit = clean ? (gMax+gMin)/2 : (gMax+gMin)/2.5;

  for (let i = 0; i < data.length; i++) {
    let isLimit = (i == 0) || (i == data.length -1);
    if (isLimit || (data[i].avgA > data[i + 1].avgA &&
  data[i].avgA > data[i - 1].avgA)) {
      if (data[i].avgA >= limit) {
        var lastIdx = locals[locals.length - 1];
        if (!lastIdx || (i - lastIdx) >= 100) {
          locals.push(i);
        }
      }
    }
  }
  return locals;
}
```

Listing 3: Function to find local maximas

Following this process results in a series of timestamps in which the local maxima can be found, in other words, timestamps that refer to roughly the same moment of a swimming stroke, as the movement is periodical. The very first and very last timestamp obtained refer to the end and start of a swimming cycle. These values are removed before proceeding to the next step.

The next step is the actual identification of stroke and lap amount. This process if very similar to the one that was previously undertaken, with the first two mentioned steps being repeated for the signal that's now been cleared of walking data. At this stage, the local maximas would've been obtained, as can be seen in Figure 4.18.

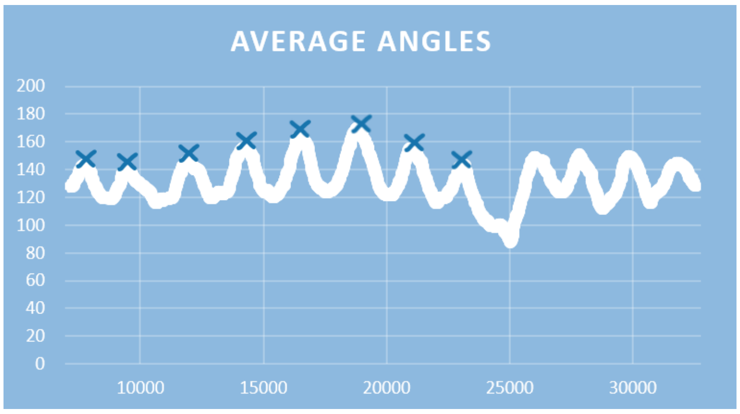


Figure 4.18: Example of maximas in a portion of the session.

After that, this process is followed:

1. **Local Minima**: The previous step (local maxima calculation) is repeated by calling the same function, but with the average angle for each time instant being inverted. This way, it is possible to estimate the "missing"peaks. In other words, since the local maxima were calculated in the previous step, this step will calculate the local minima, as shown in Figure 4.19;

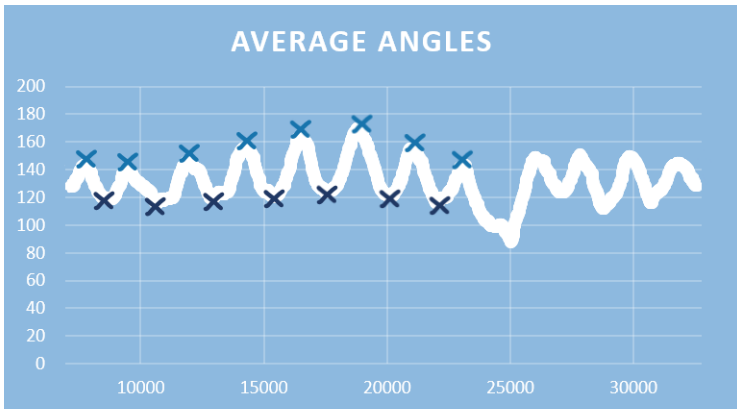


Figure 4.19: Addition of minimas in the same portion of the session.

2. **Cycle Analysis**: Using the detected maxima and minima, it is now possible to detect the strokes present in the dataset. For this, it is stipulated that a stroke will correspond to two consecutive local maxima, as long as they have a local minimum between them. Furthermore the number of instances between each maximum and its intermediate local minimum must be less than twice the average number of instances between each maximum and its adjacent minimum;

With this process completed, it's possible to not only obtain the stroke count, as explained above, but to also find an estimated lap count: these moments in the dataset are the ones that do not meet the stated requirements. Although they depend on how the turn occurs (the results obtained from turning from crawl to back is not the same and crawl to breaststroke, for example) turning points typically refer to sudden very high maximas or very low minimas appearing on the dataset, due to the velocity of the turn or locals that break the periodic cycle as they're farther from than the rest.

While the algorithm finds minimas between the maximas, the laps can thus also be easily found by comparing the values of the peaks.

Now's the time to note that a stroke is a complete movement of an arm - when dealing with breaststroke or butterfly, both arms do the movement at the same time, so this calculation is complete. However, for backstroke and freestyle, the values obtained only refer to the movement of one arm, the other one does not move at the same time. There's different opinions on what counts as a stroke in these cases: if each stroke cycle consists of two arm movements or if each time one arm completes a cycle it's another stroke. For this project, the first option was picked.

So far the number of strokes and the number of completed laps has been obtained - this last value is obtained by seeing the length of an array with the start time of every lap. With this array, it's rather simple to calculate the average lap time: one simply needs to sum the time of each lap using the starting times and dividing it by the obtained amount.

Equation 4.1 presents the calculation of the SWOLF score for each lap.

$$SWOLF = (StrokesTaken) + (SecondsTaken) \tag{4.1}$$

Regarding the SWOLF score, a structure containing each lap and it's obtained score is saved.

Finally, as there's currently not a more robust way of doing this, the swum distance is calculated by multiplying the pool size by the number of laps - this can clearly not be a perfect indicator, but its a close approximation.

Once a file is downloaded, the `getSessionInfo()` is called, sending the data received. It returns a structure with all of the pertinent results to ease accessibility. The JSON version of this structure is saved in the treated files bucket. In Chapter 5 this process will be further validated.

4.3.8 Style Identification

Unfortunately, since the sensor module is not ready to be used in the water, the study of the differentiation between swimming styles could not be properly conducted. Not having a consistent, trained swimmer to obtain reference values for the various styles hindered this part of the project. Nevertheless, the it is still prepared to allow the display of the swimming styles in future work, as soon as reference values can be obtained. This section will explain how.

The structure of swimming information already contains an entry regarding the swum style(s) in each lap. For now, the project will be assuming the swum style is freestyle.

In the future, all that's necessary is to call a function that will analyze the data and, from the reference values and parameters, return the styles swum and the

So how could this function work? Just how distinguishable are the signals obtained?

Let's take a look at graphs obtained for distinct styles. Figure 4.20 shows the roll and pitch signal difference for breaststroke and freestyle (the yaw value doesn't vary enough between styles to matter).

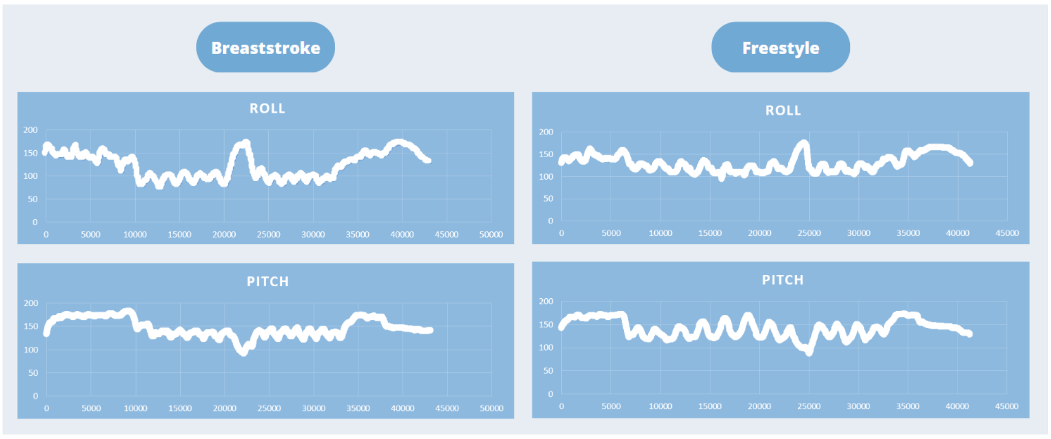


Figure 4.20: Comparison of pitch and roll for freestyle and breaststroke.

The graphs clearly display different patterns in the signal. Focusing on the freestyle style first, the swimmer positions themselves belly-down and uses their arms as a “padle”to propel themselves forward, results in the following outcomes: In this type of swimming, there is a greater amplitude in pitch results compared to those of roll - the rotation can be described as being more more "up and down"than laterally around oneself. On the other hand, in breaststroke, the swimmer's torso does not rotate and arms are used to "pull"oneself forward. The graphs exhibits a greater range of roll compared to pitch, as the arm does not effectively lift the sensor module above itself as much, but rolls around oneself when going back.

Similar results can be found for butterfly and backstroke - although these are not as accurate in the current state, without using the sensor in the pool. Regardless, in the butterfly style yaw and roll values will remain mostly relatively constant, as the biggest difference will be seen in the pitch signal and the arms go up and down. Backstroke is the style where all axis have considerable variation, since the arm rotate in different ways along the move - pitch will be higher as the arm is moving to the back of the head, before reaching the water, and roll and yaw will noticeably vary as the arm goes sideways back to the tail of the body, in the water.

Simple parameters can thus be used for style distinction. For example, the following attributes could be used:

- **Average in one or more of the axis** - for example, butterfly will have a higher amplitude in pitch values;
- Similarly, the **standard deviation** can also be used;
- **Average distance between peaks** - not only do the styles have slightly different rhythms, but since the sensor is only used in one arm styles where both arms move at the same time will have shortest distances between peaks/strokes;

Furthermore, as the focus is analysis of the stroke cycles, this process might require previous data treatment in order to smooth the curves obtained.

Once again, without a consistent well-trained swimmer, trying to find a good parametrization to distinguish the styles would only result in a solution that's too tailor-fitted for the tester - a tester that would be a non-swimmer member of the group.

So the focus was then shifted towards laying the foundation to allow these values to be displayed easily, once they can be effectively obtained.

The donut chart on the main page is filled based on the structure with the session analysis - whether the style is seen per lap or in the overall scope of the session (meaning more than one style is allowed per lap). The system sums the amount of time a style was trained for and divides it by the total swimming times, this results in the percentage of time that style was trained for. The chart receives these percentages and adapts accordingly.

Chapter 5

Validation and Testing

This chapter addresses the testing process and results obtained using the proposed system.

First, the obtaining of values on the sensor module will be put to a test in Section 5.1. Once this is proved for both basic movements and swimming movements, the mobile app will be tested as described in Section 5.2 to make sure sessions can successfully start and end via Bluetooth communication. In Section 5.3 the stroke count will be verified for the various swimming styles.

Lastly, the entire analysis system will be validated by showing the swimming session statistic results (Section 5.4) and the session camparision/ranking page (Section 5.5).

5.1 Validation of the Angle Values Obtainment

The purpose of this first test was to guarantee that in the sensor module the angles are correctly read.

The sensor was placed on a flat surface, and then rotated by 90º back and forth. The validation of the roll values will be done first with a more in depth explanation in order to better explain the obtained results.

It's important to first refer what results will be expected: the signal obtained will graph the roll value by timestamp, meaning that the roll value should remain constant until the sensor module is rotated.

In Figure 5.1 an exemplification of the movement axis and of the results obtained can be seen.

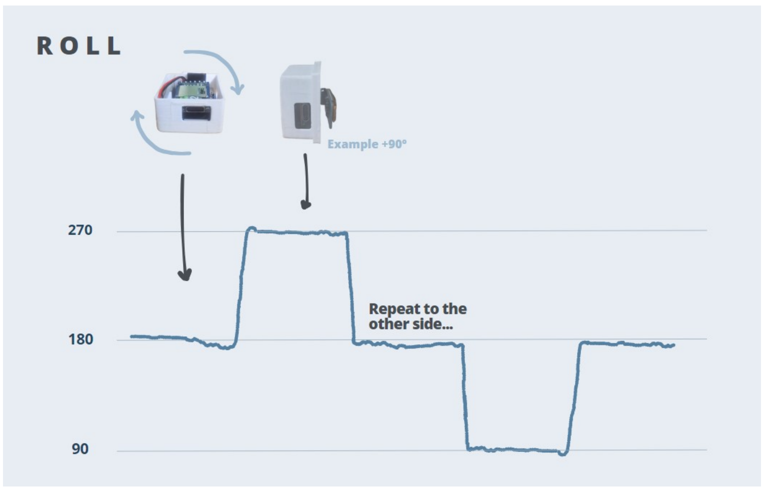


Figure 5.1: Roll values obtained.

The results obtained match the prediction previously made. The roll values are obtained within the 0º to 270º range, starting at 180º and displaying sudden changes of 90º.

Now, the pitch and yaw rotations should also be validated. The results can be seen in Figure 5.2.

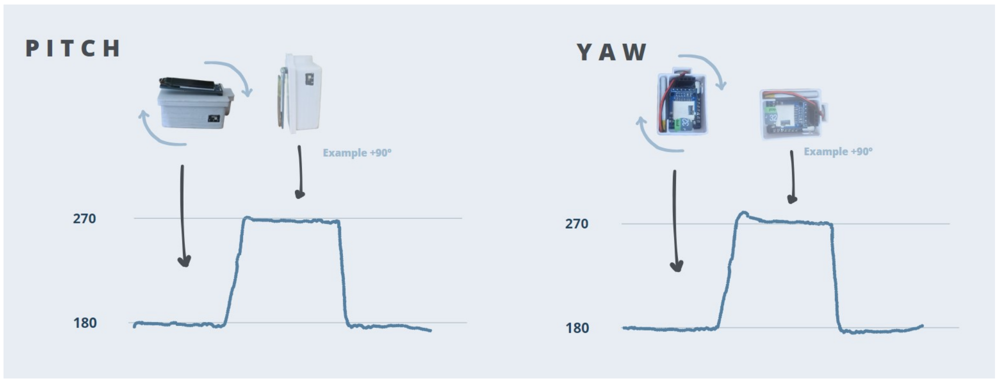


Figure 5.2: Pitch and yaw values obtained.

After this process, it is possible to verify the correct acquisition of angles in controlled movements in each axis of the orthogonal reference frame. This test gives the “green light”to further evaluate the module’s behaviour when used during an actual swimming stroke.

Figure 5.3 shows the roll and pitch obtained when simulation freestyle swimming - the yaw is not as clear as it does not discriminate this style as much.

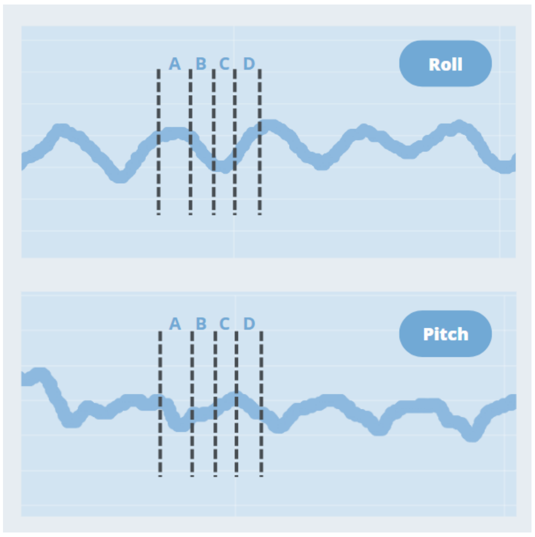


Figure 5.3: Freestyle “stages”detected.

It’s necessary to quickly review how the freestyle swimming style works, to better understand the most important phases that should be visible on the obtained signals.

Freestyle swimming, also known as front crawl, involves a coordinated movement of the arms and legs to propel the swimmer forward. The stroke can be divided into four main stages: the catch, the pull, the recovery, and the kick. Based on the graph, the following moments can be found:

- **A - Recovery:** During the recovery phase, the arm moves out of the water, preparing for the next stroke cycle.
- **B - Catch:** Begins when the hand reenters the water in front of the swimmer’s head.
- **C - Pull:** As the hand and forearm catch the water, the swimmer pulls the arm back in a semicircular motion, exerting force against the water. It is the main propulsive phase of the stroke, generating forward momentum.

- **D - Propulsion** Provides additional propulsion and helps maintain balance in the water, the arm finishes it's pull through the water before leaving, while the legs make a kicking motion.

As the graphs shown, cyclical movements are visible, each cycle symbolizing a arm movement. The curves can be further divided into different sections - while it either remains roughly at the same values, increases or decreases.

5.2 Mobile Connection and Communication

Regarding the confirmation of the mobile app application implementation (as described in Section 4.2), two factors should be tested: if it manages to detect the sensor module's BLE central and connect to it, and if it truly manages to deploy and end a training session.

Upon launching the app, the first screen seen in the Bluetooth connection screen. In the background, continuously, the application is searching for BLE centrals to connect to. If at the start the sensor module is turned off, nothing should be found (as there are no more SWIMU devices nearby). Once it's turned on - something that's visible because a green light turns on for debugging purposes - it should appear immediately without the need to refresh the page. Figure 5.4 shows the result of this sequence.

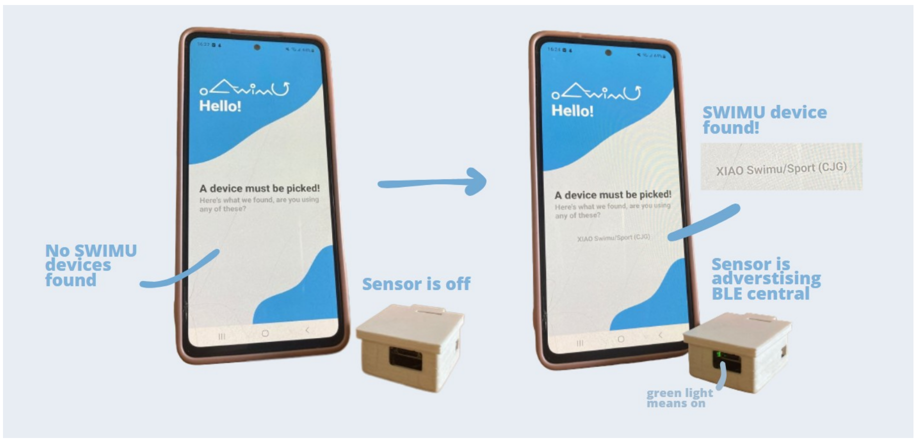


Figure 5.4: Mobile bluetooth screen detecting a device.

The found device did indeed pop up on the list immediately, as soon as the microcontroller was turned on - since it starts advertising the central almost instantaneously, and the search for devices is also constant the waiting time is reduced to the minimum.

At this stage, before transferring the files in order to see if sessions are saved correctly, the only way to truly test the reception of the timestamp and set distance is by using an LED for debugging motives once again.

Once the sensor module starts saving data (meaning all values were successfully received) a blue LED light is turned on. Figure 5.5 shows the color obtained.

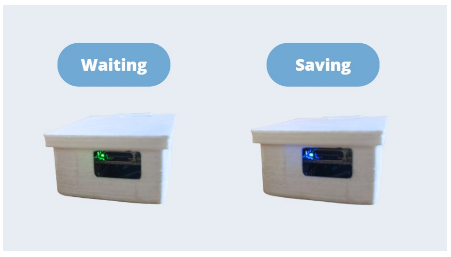


Figure 5.5: Lights changing depending on the sensor module's state.

The color change happens as soon as the user presses the start button, confirming the correct sending of the values.

Furthermore, the user can leave the app without stopping the session even if it disconnects from the central. As mentioned in Section 4.1.7 the file the sensor is using stays as a global variable and only closes when another Current Time Characteristic is received, stopping the session and moving on to the waiting state once more.

5.3 Stroke Calculation

This section aims to show the results obtained from applying the stroke count analysis process described in Section 4.3.7, before further exploring the remaining calculations.

From this point forward, there are some very important points to me made:

- This simulation is not done on the water, as the sensor module is not prepared for that, yet. Due to this, values may vary a little from the end result as the water offers more resistence to the arm, making it impossible to fully simulate the speed at which the arm would move;

- These tests were not simulated by a trained or knowledgeable swimmer, but by a member of the group;

For this test, 50 strokes will be simulated on a single lap for the various styles. Table 5.1 shows 8 different results obtained when testing the system with a freestyle single-lapped simulation.

Table 5.1: Study of the results for 50 strokes - freestyle.

Test Number	Stroke Count Obtained	Deviation
1	48	2
2	50	0
3	47	3
4	47	3
5	50	0
6	50	0
7	51	1
8	48	2
	Average Deviation	1.38

Out of 8 tests, 3 of them were exact on the estimated stroke count, the remaining tests were off by, at most, 3 strokes. On average, the stroke count for freestyle deviates, in average, 1.38 strokes in 50 strokes.

Now, it's time to do a varied number of strokes, and see if this deviation escalates the longer the session is.

Table 5.2 shows the results obtained from tests with 20, 50, 70 and 100 strokes, with 2 repeats each

Table 5.2: Study of the results for varying number of strokes - freestyle.

Real Strokes Count	Average Stroke Count Obtained	Average Deviation
50	48.5	1.5
70	69.5	0.5
90	88.5	1.5
100	98	2

This test helped to conclude that, at least for one lap, the average deviation do not over escalate the bigger the number of laps is. In other words, it doesn't seem to be proportional to the number of strokes - the deviation might then have more to do with how much movement happens before and after the session actually starts and how much of this noise is removed from the data than gets analyzed.

Lastly similar tests can be done for the other styles. This time, 5 sessions of each style were simulated, with 20 strokes each. The results appear in Table 5.3.

Table 5.3: Study of the number of strokes variation.

Style	Average Deviation
Freestyle	0.8
Backstroke	0.9
Breaststroke	0
Butterfly	0

To get better results a bigger amount of samples should be used - but these tests will have to be repeated once the sensor is prepared to enter the water. For now, this amount is enough to provide some insight in the functionality.

With this small sample amount, it can be concluded that for both butterfly and breaststroke the results were exact - all five sessions resulted in the exact amount of strokes executed. The remaining styles presented small variations.

Any shifts in values can be due to calibration problems or also simply by the lack of consistency on the testing strokes - the tester was not a trained swimmer.

5.4 Full Session Testing

In this section tests for sessions with more than one lap are presented. It aims to show how accurate the method is for stroke and lap count, distance estimation, SWOLF score calculation and time calculations;

Section 5.4.1 only one style will be swum, in order to validate the most basic type of sessions that can be trained. Then, in Section 5.4.2 more than one style will be swum, to make sure that doesn't disturb the acquisition of any values. The goal is to now verify if every displayed value in calculated correctly.

5.4.1 Same Style Every Lap

A session was simulated with the characteristics shown in Table 5.4.

Table 5.4: Test Session 1.

Lap	Style	Number of Strokes
1	Freestyle	10
2	Freestyle	10
3	Freestyle	10
4	Freestyle	10

The "pool"size sent was 20m (although this is no more than a simulation out of water). All laps were swum with the same number of strokes and same style. The time it took to complete was calculated using a chronometer, resulting in roughly 2 minutes and 20 seconds. Upon transferring the file, the page in Figure 5.6 was presented:

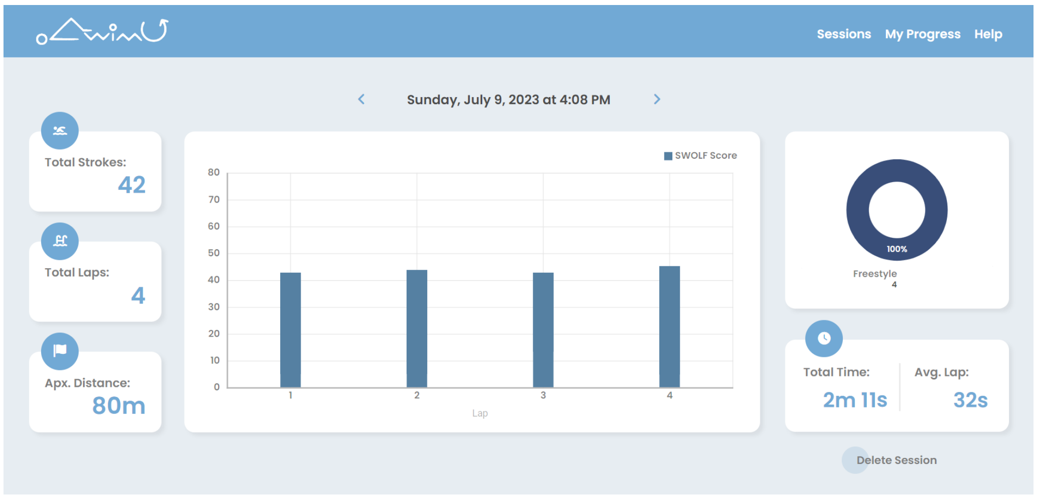


Figure 5.6: Results of the first session.

Let's now think about the values expected and the values obtained. This comparison is made in Table 5.5.

Table 5.5: Test Session 1 results.

Value	Expected	Obtained
Number of Strokes	40	42
Number of Laps	4	4
Total Time	2m 15s	2m 11s
Average Time	32.7	32

The number of laps is perfectly acquired. Slight variations can be seen in the times obtained - this may be due to the “cleaning”of the non-swimming sections of the session before data is calculated. This causes slight variations in the time cut and, depending on where the data is cute, may be causing the variations in stroke count, as well - by seeing the results obtained in the JSON it was concluded that only the first and last lap has one extra stroke calculated, which is exactly where data is cleaned.

Regarding the SWOLF, the values obtained were all correct based on the stroke and lap time. These scores were, respectively 42, 42, 43 and 45.

Admittedly, during the simulation and even a typical practice, it's normal for the arms to start getting more tired by the end, especially if the swimmer isn't too proficient - this happened and was reflected on the SWOLF score, the stroke cycle frequency decreased and thus increased the time it took to complete a lap, also increasing the SWOLF score.

5.4.2 Varying Styles

Table 5.6 displays the distribution of a session with more laps while varying stroke amount and swum style.

Table 5.6: Test Session 2.

Lap	Style	Number of Strokes
1 and 2	Freestyle	15
3 and 4	Backstroke	20
5 and 6	Breaststroke	10
7 and 8	Butterfly	25
9 and 10	Freestyle	20

Figure 5.7 shows the results of the tests conducted to confirm the stroke and SWOLF calculations while varying styles and number of strokes.

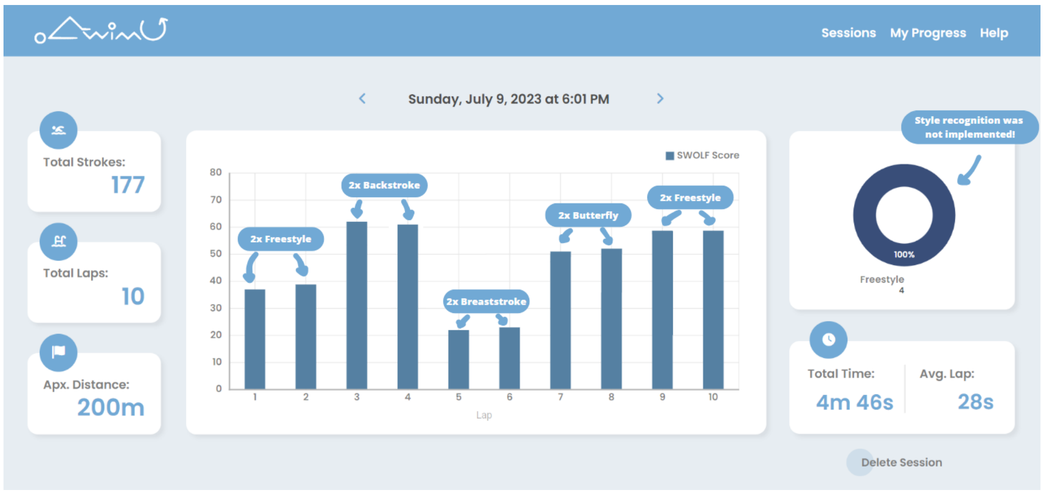


Figure 5.7: Results of the second session.

Once again, the lap count got the correct count - at this stage, it seems like the implementation is not “stuck”to a certain swimming style - even by varying the style all values seem to be correctly calculated.

Regarding the SWOLF, the values obtained are clearly not accurate to those that would be obtained in a real session in a pool - a pool is always the same length, doing different numbers of strokes in the same style per lap and taking it as a single round is a liberty taken to put the system to the test.

With this in mind, the results perfectly match the ones that could be expected: lap number 5 and 6 have a much lower SWOLF score because the number of strokes is also much smaller.

With these two tests, a first confirmation of the data analysis module is completed, and results obtained seem to always translate a correct lap and SWOLF count - the number of strokes might suffer from slight variations, but the remaining values adapt to this behaviour.

One last thing that should be underlined in this chapter is, once again, the fact that these tests were not made by a trained swimmer nor were they made inside a pool - these results and study help lay the foundations for how all these values can be obtained, but once put in practice in a real situation, parameters might need to be further corrected as to have a more generic system.

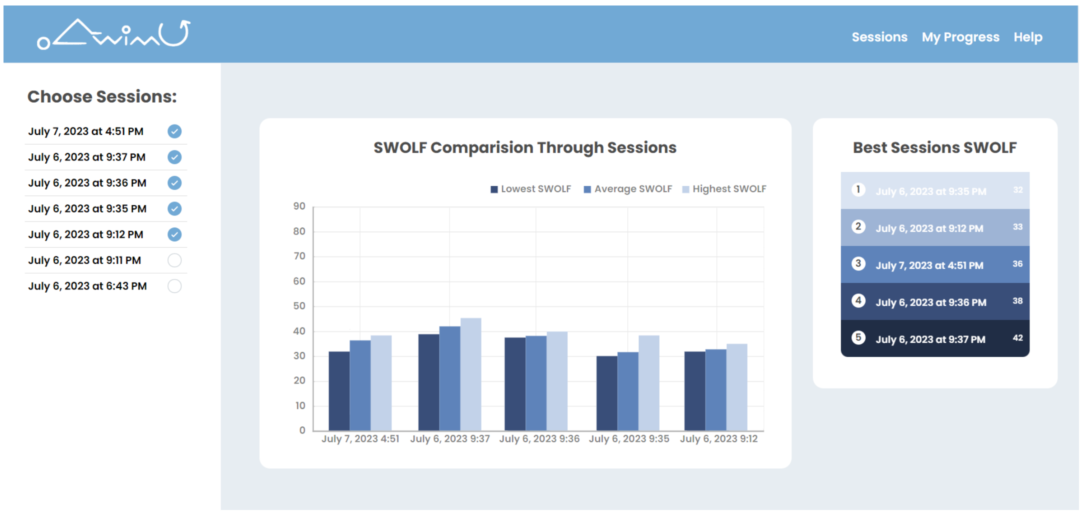
5.5 Session Comparison

The last remaining confirmation that needs to be made is if the session comparison/ranking is working correctly. This will be a basic validation that serves as proof that the entire application in functioning correctly.

Typically, users would pick similar sessions to compare in this page: for now, as the focus is to conclude the correct ranking, this section will focus solely on confirming if the ranking is done as correctly, based on the average SWOLF seen on the graph.

This average SWOLF is also obtained from the JSON version of the sessions saved in the Cloud storage.

Simply taking a look at the comparison page is enough to verify it's functionality. Let's focus on Figure 5.8.



Choose Sessions:

- July 7, 2023 at 4:51 PM ✓
- July 6, 2023 at 9:37 PM ✓
- July 6, 2023 at 9:36 PM ✓
- July 6, 2023 at 9:35 PM ✓
- July 6, 2023 at 9:12 PM ✓
- July 6, 2023 at 9:11 PM
- July 6, 2023 at 6:43 PM

Best Sessions SWOLF

1	July 6, 2023 at 9:35 PM	32
2	July 6, 2023 at 9:12 PM	33
3	July 7, 2023 at 4:51 PM	36
4	July 6, 2023 at 9:36 PM	38
5	July 6, 2023 at 9:37 PM	42

Figure 5.8: Functioning comparison page.

Looking at the bar chart and at the ranking table, it's clear that the ranking is working accordingly to to the size of this bar.

Hovering over the bars details the value each one represents. This page will allow the users to see how consistent their swimming has been during the selected sessions.

Chapter 6

Conclusions and Future Work

Upon reaching the culmination of this project, it is essential to take a moment and reflect upon the undertaken task and whether or not the initial proposed requirements were met.

The objective of this project was the development of a system capable of assisting athletes, with varying levels of proficiency, in the practice of swimming. Upon identifying that in spite of being one of the oldest activities, there is a technological lag regarding swimming practices when compared to other sports. Although there is a range of wearable devices such as smartwatches, smartbands, or swimming goggles capable of performing analysis based on inertial signals in swimming practice, these devices often tend to not only be very expensive due to the requirement of being water-proof, but also limit access to the collected information, and are unable to detect errors in the execution of different strokes. Based on this premise, a low-cost and open-source inertial signal analysis system was created last year by Gabriel Diaz, incorporating both hardware and software components.

Taking on the responsibility of continuing of Diaz' work, the development of the project presented in this report begun by following a similar process.

The implementation started with the programming of the Seeed Studio XIAO nRF52840 (Sense) microcontroller in order to create an entity capable of collecting data from the calibrated IMU when so required, and of transmitting said data to the web and mobile external applications. The implementation of a state machine allowed for a methodology that's easier to understand and less error prone (less actions are taking place at the same time).

Furthermore, both the developed web and mobile application manage to communicate with the sensor module following the implemented protocols, and their implementation ensures support for multiple platforms, thanks to the use of React Native as programming language, for example.

Regarding the visual side, the web application's was developed with the intention of creating a user-friendly interface that can be easily understood by swimmers of all skill levels alike, and that provides all the essential metrics that might be of interest at this stage.

Lastly, after testing the system, especially when it came to data analysis, the system obtained very positive results. It is capturing and analyzing the relevant data accurately, providing valuable insights to the swimmer. All in all, the results imply that the built system is appropriate and well-designed, confirming that it is correctly identifying stroke counts, lap counts, and correctly doing related calculations, such as the SWOLF score.

The overall architecture keeps the user in mind, from the use of asynchronous functions to enhance responsiveness and user experience, to making sure the waiting times are as minimized as possible, hoping to build the a system as effective as possible.

To sum it up, the goals and requirements that were set in the beginning were successfully fulfilled, and resulted in three functional components that follow the initially established architecture. All the code is open-source, available for anyone to see.

6.1 Future Work

While this project may be concluding, the journey does not end here. The accomplishments achieved lay the foundation for future projects and extensions - just as this carried on from Diaz's original project.

Let's delve into the potential paths that can be pursued henceforth.

- **Improvement of Communication Processes:** As was mentioned, it was not possible to undertake the task of implementing training session via Bluetooth. The future development of such a Bluetooth protocol would provide a better user experience by increasing the download speed and removing the necessity of cables.

- **More Training Statistics:** Integration of advanced tracking and analysis capabilities to provide users with even more comprehensive training statistics. This could include features like a better estimation of covered distance using more precise methods or the calculation of average speed. This requires changes to the sensor module used.

- **Planification of Swimming Sessions:** The sensor module is equipped with a motor. It would be interesting to allow users to enter in the application the styles they'd like to practice. After sending this data to the module, the motor would vibrate to indicate it's time to switch to another style. Even better, users could state their goals for an upcoming competition, for example, and have a personalized training plan that worked similarly.

- **Personalized Training Plans:** Building up on training plans, it would be interesting to implement a feature that allows users to create personalized training plans based on different factors - their fitness goals, abilities, and time constraints, for example. The application should then offer flexibility in selecting training schedules, exercises, and intensity levels, catering to both beginners and advanced athletes.

- **Gamification Elements:** Athletes are inherently competitive, and constantly want to best themselves. An important feature to make the training experience more engaging and motivating would be the inclusion of badges or rewards for completing certain milestones, participating in challenges, or achieving personal bests. Leaderboards and friendly competitions can also be included to encourage healthy competition among users.

By incorporating these features the project will benefit from the touch to the user experience, making it more interactive, and enjoyable.

Besides this, although this project focused on studying statistics applied to swimming, it is possible to further extend the solution to include a wider range of sports. The sensor module and the application implementation do not need to be changed to do this - the way they both function is generalized in a way that is fit for all kinds of sports. The changes would only need to occur on the web application. The data analysis module in the web application could be modified to calculate statistics pertinent to other types of sports, returning a structure with this information. Then, the graphs in the browser can be changed or adapted to fit these new values. This way, the user could be able to select their preferred sport, such as soccer, basketball or cycling, and more.

All in all, and as a last note, one of the main takeaways from this project is just how diverse the possibilities from here on out are. The possible future outcomes and achievements above mentioned highlighted the immense potential for growth and innovation, and are only a portion of the wide range of choices that truly exist.

The groundwork laid by this project provides a solid framework upon which future initiatives and projects can be built.

Appendix A

Board and Circuit Details

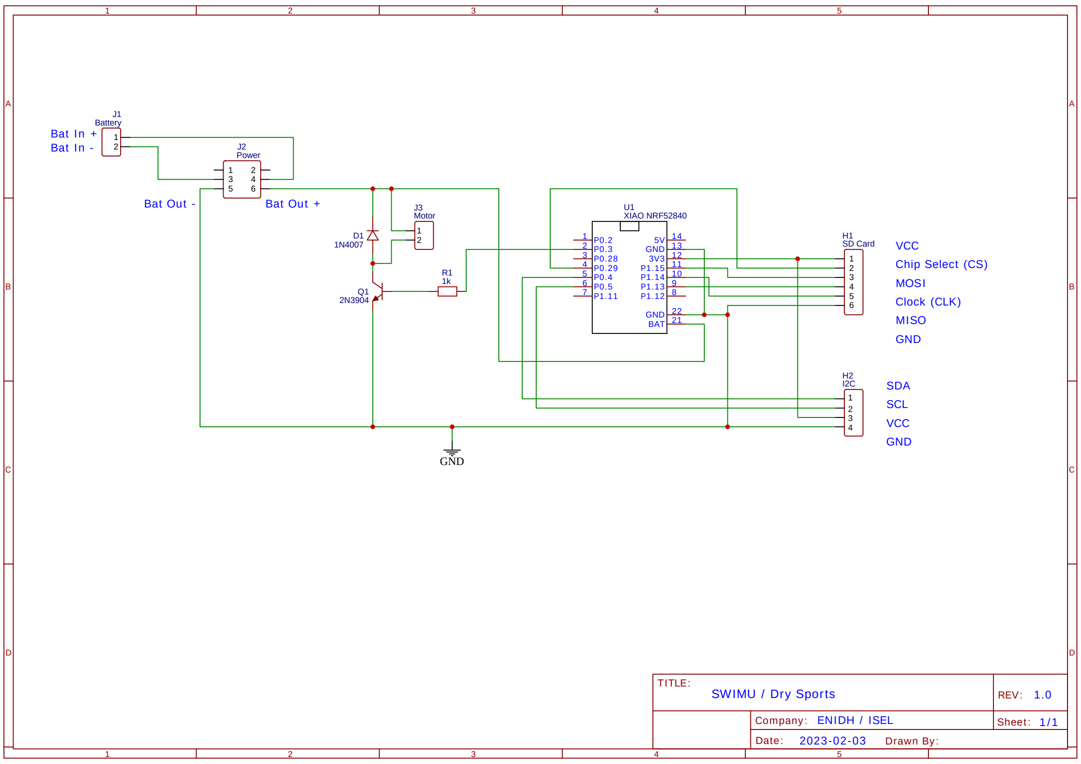


Figure A.1: Sensor Schematic.

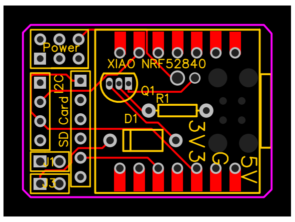


Figure A.2: Printed Circuit Board - Top View.

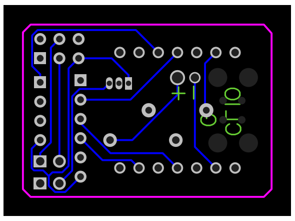


Figure A.3: Printed Circuit Board - Bottom View.

Appendix B

Web Application Schema

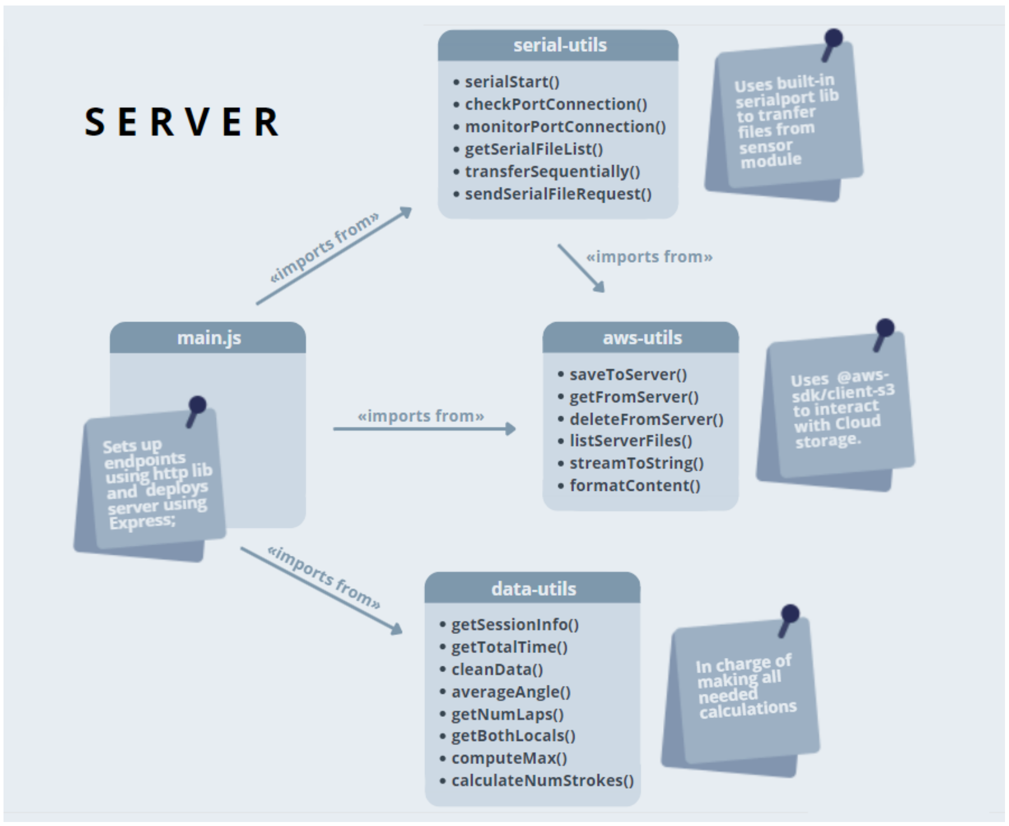


Figure B.1: Server schema.

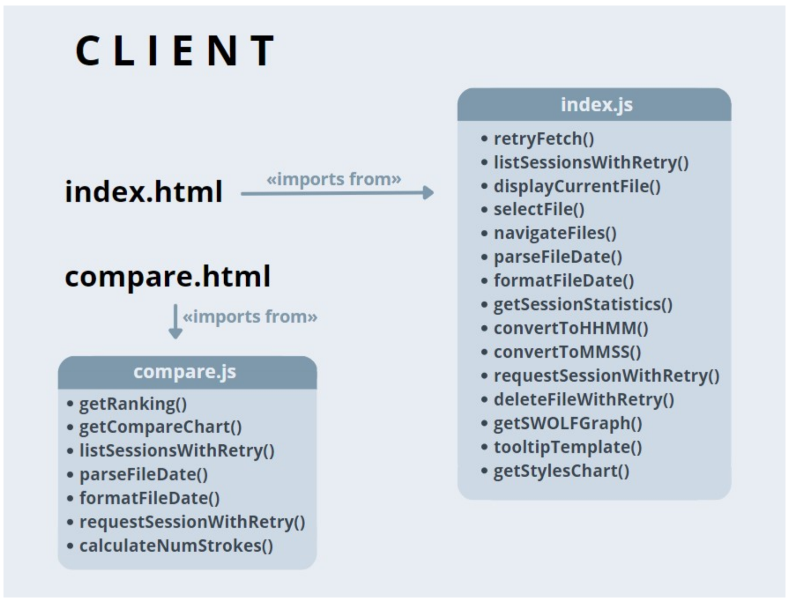


Figure B.2: Client schema.

References

[Aliexpress, 2023] Aliexpress (2023). Micro sd tf cartão de memória. https://campaign.aliexpress.com/item/1005005514934667.html. Last acessed on 11.07.2023.

[Amazon, 2023] Amazon (2023). Amazon cloud service. https://aws.amazon.com/pt/s3/. Last acessed on 11.07.2023.

[Apple, 2023a] Apple (2023a). Apple - specs. https://support.apple.com/en-us/HT209393,. Last acessed on 11.07.2023.

[Apple, 2023b] Apple (2023b). Apple watch - website. https://www.apple.com/pt/watch/. Last acessed on 11.07.2023.

[Arduino, 2023] Arduino (2023). About the nano rp2040 connect. https://docs.arduino.cc/hardware/nano-rp2040-connect. Last acessed on 11.07.2023.

[DHTMLX, 2023] DHTMLX (2023). Dhtmlx library. https://dhtmlx.com/docs/products/dhtmlxGantt/?gad=1&gclid=CjwKCAjw2K6lBhBXEiwA5RjtCcupHCtYvNKgheHxP739mEjB36kjlytf2484uUm2A8GWtXIyrgjgfhoCQ8 BwE. Last acessed on 11.07.2023.

[Diaz, 2022] Diaz, G. H. P. (December 2022). Aprendizagem automática de estilos de natação - swimu. Master's thesis, ISEL.

[EasyEDA, 2023] EasyEDA (2023). easyeda - getting started. https://docs.easyeda.com/en/FAQ/Editor/index.html. Last acessed on 11.07.2023.

[Express.js, 2023] Express.js (2023). Express - node.js web application framework. https://expressjs.com/. Last acessed on 11.07.2023.

[Fitbit, 2023a] Fitbit (2023a). Fitbit - specs. https://www.fitbit.com/global/us/products/smartwatches/versa4?sku=523BKBK,. Last acessed on 11.07.2023.

[Fitbit, 2023b] Fitbit (2023b). Fitbit - website. https://www.fitbit.com/global/us/home. Last acessed on 11.07.2023.

[Garmin, 2023a] Garmin (2023a). Garmin - about garmin. https://www.garmin.com/en-US/company/about-garmin/. Last acessed on 11.07.2023.

[Garmin, 2023b] Garmin (2023b). Garmin - specs. https://www.garmin.com/en-US/p/665374#specs,. Last acessed on 11.07.2023.

[Github, 2018] Github (2018). React native waveview. https://github.com/CubeSugar/react-native-waveview. Last acessed on 11.07.2023.

[Github, 2021] Github (2021). React native ble plx. https://github.com/dotintent/react-native-ble-plx. Last acessed on 11.07.2023.

[jlcpcb, 2023] jlcpcb (2023). Jlpcb website. https://jlcpcb.com/. Last acessed on 11.07.2023.

[Letícia Lucas, Daniela Levezinho, 2023] Letícia Lucas, Daniela Levezinho (2023). Project repository. https://github.com/DanielaLevezinho/SWIMU. Last accessed on 11.07.2023.

[Little Fishies, 2021] Little Fishies (2021). Different swimming style and strokes swimming style. https://swimlessons.sg/different-swimming-style-strokes/. Last accessed on 11.07.2023.

[Mauser, 2023] Mauser (2023). Bateria 3.7v 400mah. https://mauser.pt/catalog/product_info.php?products_id=035-9004. Last acessed on 11.07.2023.

[Node.js, 2023] Node.js (2023). Node.js - introduction. https://nodejs.dev/pt/learn/. Last acessed on 11.07.2023.

[NovelBit, 2023] NovelBit (2023). Bluetooth - maximun-throughput. https://novelbits.io/bluetooth-5-speed-maximum-throughput/,. Last acessed on 11.07.2023.

[React Native, 2021] React Native (2021). React native - getting started. https://reactnative.dev/docs/getting-started. Last acessed on 11.07.2023.

[Seeed, 2023] Seeed (2023). Microcontrolador seeed studio xiao nrf52840 sense. https://www.seeedstudio.com/Seeed-XIAO-BLE-Sense-nRF52840-p-5253.html. Last acessed on 11.07.2023.

[Seeed Studio, 2023] Seeed Studio (2023). Getting started with seeed studio xiao nrf52840 (sense). https://wiki.seeedstudio.com/XIAO_BLE/. Last acessed on 11.07.2023.

[Worten, 2023] Worten (2023). Cartão de memória micro sd pny elite 32 gb. https://www.worten.pt/produtos/cartao-de-memoria-micro-sd-pny-elite-32-gb-6985171. Last acessed on 11.07.2023.