INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Smart Fitness System: Training Programming

PEDRO NUNO TEIXEIRA LOPES REIS GONÇALVES
Outubro de 2019

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

Smart Fitness System

Training Programming

Pedro Nuno Teixeira Lopes dos Reis Gonçalves

Dissertation for obtaining a Master's Degree in Informatics Engineering, Area of Specialization in Software Engineering

Supervisor: Luiz Felipe Rocha de Faria

Co-supervisor: António Constantino Lopes Martins

Júri:

Vogais:

Porto, October 2019

Resumo

Sistemas de recomendação no geral estão a ser cada vez mais usados por empresas que procuram oferecer uma experiência de utilização mais individual e personalizada aos seus clientes. Obter *feedback* em transações de negócio online nunca foi tão fácil e acessível, o que apenas ajuda a catalisar a evolução dos sistemas de recomendação.

Adicionalmente, o uso de dispositivos tecnológicos como *smartphones* e computadores, juntamente com a conexão à internet, estão também a crescer a um ritmo acelerado sem sinal de paragem em vista. Juntando-se a este grupo de indústrias em crescimento está a indústria *fitness*, que está a ficar cada vez mais popular. Com isto, mais e mais pessoas estão a começar a usar os dispositivos mencionados anteriormente em combinação com as suas atividades *fitness*, para aumentar o seu desempenho, monitorizar progresso, definir objetivos, entre outros. Consequentemente, o mercado para sistemas *fitness* (p.e. aplicações *fitness*) está a aumentar e já é bastante denso. No entanto, a qualidade associada com tais sistemas fica um pouco aquém tanto em termos de inovação como de funcionalidades essenciais.

Como resultado disto, este projeto propôs uma solução – um sistema *fitness* sob a forma de uma aplicação móvel aliada a um poderoso sistema de recomendação. Este sistema é pretendido que providencie uma experiência mais individual e personalizada para qualquer tipo de utilizador *fitness* através da oferta de funcionalidades essenciais como registo e monitorização de informação, análise de progresso, e também através de funcionalidades inovadoras como a implementação de um sistema de recomendação capaz de sugerir tópicos relacionados com *fitness* (p.e. regimes de treino ou exercícios específicos) baseado em múltiplos fatores como os objetivos, características individuais e historial de cada utilizador. Além do mais, deve também oferecer um assistente pessoal virtual, onde os utilizadores podem expressar as suas questões e dúvidas, e tê-las respondidas instantaneamente por um *chatbot*.

Durante o desenvolvimento foi decidido que um segundo sistema de recomendação seria necessário para melhorar o sistema no geral. Este, o sistema, depois de implementado, foi avaliado e pode ser concluído que o resultado foi um sucesso, tendo passado em todas as métricas definidas, exceto uma, com classificações médias nos questionários de satisfação acima de 4/5. O feedback obtido por um especialista no sistema de recomendação foi altamente vantajoso e no geral decentemente positivo, apenas com algumas questões que necessitam de melhoramento. Embora o sistema de recomendação inteligente não tenha conseguido ser testado com informação aplicável, a investigação e trabalho feito constituem uma mais valia caso mais tarde exista a possibilidade de aplicar dados reais.

Palavras-chave: *Fitness*, Sistemas de recomendação, Aplicação móvel, *Chatbot*

Abstract

Recommender systems in general are increasingly becoming more exploited by companies who seek to provide a more individual and personalized user-experience to their customers. The fact that providing feedback on online business transactions is also becoming ever so easier only helps to catalyze the evolution of recommender systems.

Moreover, the use of technological devices such as smartphones and computers, in conjunction with an internet connection, are also continuing to grow at a fast pace, with no slowing down in sight. Joining on this group of growing industries is the fitness sector, which is becoming increasingly popular. With this, more and more people are starting to use the aforementioned devices in combination with their fitness activities, to boost performance, monitor progress, define goals, among other things. Consequently, the market for fitness systems (i.e. fitness applications) is expanding and is already very dense. However, the associated quality with such systems falls short both in terms of innovation and even crucial features.

As a result, this dissertation proposes a solution - an innovative fitness system in the form of a mobile application allied with a powerful recommender system. The system is intended to provide a more individual and personalized experience to any type of fitness user through the offering of crucial features including the log and monitor of information, progress analysis, and also through innovative features such as the implementation of a recommender system capable of making fitness-related suggestions (i.e. training regimens or specific exercises) based on multiple factors like the user's individual goals, characteristics, and history. Additionally, it should also provide a personal virtual assistant, where users can express their questions and doubts and have them answered instantly by a *chatbot*.

During development, it was decided that a second recommender system was required to improve the system as a whole. This, the system, after being implemented, was evaluated and it can be concluded that the result was a success, having passed in all the defined metrics, except one, with average classifications of 4/5 on the satisfaction inquiries. The feedback obtained from the expert on the recommender system was highly useful and, in general, decently positive, having only a few questions that need improvement. Even though the intelligent recommender system couldn't be tested with applicable data, the investigation and work done constitute a great asset in case there's the opportunity to employ real data.

Keywords: Fitness, Recommender systems, Mobile app, Chatbot

Index

Figures Index

Tables Index

Acronyms

AI	*Artificial Intelligence*
PVA	*Personal Virtual Assistant*
RPE	*Rated Perceived Exertion*
1RM	*1-Rep Max*
MF	*Matrix Factorization*
CF	*Collaborative Filtering*
UB-CF	*User-Based Collaborative Filtering*
IB-CF	*Item-Based Collaborative Filtering*
SVD	*Singular Value Decomposition*
HIIT	*High Intensity Interval Training*
DTO	*Data Transfer Object*
LINQ	*Language Integrated Query*
UI	*User Interface*
LUIS	*Language Understanding Intelligent Service*

1 Introduction

This chapter is dedicated to the presentation and introduction of the present project. Firstly, by contextualizing it, establishing the problem associated with it, the goals set to achieve, the motivation behind it and also the methodology employed. Lastly, the document's structure is defined and explained in order to provide a more pleasing experience to the reader.

1.1 Context

The fitness industry is experiencing a continuous growth both in terms of adherents and total revenue. The reasons behind such growth can have multiple answers, but one of them, and likely one of the most relevant ones is its coupling with technology. More specifically, wearables, make it very convenient for people to monitor their digital health feedback, allowing them to pay more attention and make more healthy decisions.

Being healthy and fit obviously carries tremendous advantages for people, both physically and mentally. This, allied to the fact that the whole fitness and mobile industry are growing, now more than ever, only makes it desirable to combine them in a distinctive manner, through a fitness mobile application.

Given that the project is being developed in conjunction with another author, this report will be directed towards the documentation and implementation of the training Programs aspect of the system. There are numerous concepts and predefinitions that need to be exposed, in order to clearly contextualize the present project. This, and more information relative to the contextualization of the said project can be found in a later Context section.

1.2 Problem

According to *AGAP* (*Barómetro da Associação dos Ginásios de Portugal*) [1], in 2015, the Portuguese fitness market grew 13%, totaling approximately 730 thousand people, or 7.1% of the population [2]. Additionally, researches focused on other countries (i.e. Chile, Germany and France), concluded the same, the fitness industry suffered a huge growth. [3] [4] It is obvious, then, that the fitness culture is "trending" and nearly every prediction indicates that. [4]

Given that it is crucial to maintain some sort of structured training Program in order to evolve and have positive changes [5], and because this is usually done by personal trainers, it is also important that the users themselves conduct some sort of monitoring. In many cases, in more recreative individuals, this monitoring is deficient or non-existent, directly affecting their results [6], leading to demotivation or even waiver. On the other hand, in more advanced users that possess well-structured and monitored training Programs, there are other kinds of issues, suchlike the difficulty and complexity of the monitorization and the need to extract useful, relevant and personalized information out of the results. [7] In both cases, the unavailability of personal trainers doesn't allow for an effective monitoring of all practitioners, leading to inadequate and inefficient practices to each one's needs. [8]

1.3 Goals

The main goal of the present dissertation is to develop an individualized and personalized monitoring and planning fitness system, allied with a recommender system adapted to each user's profile and specific needs that assists them in their fitness journey. This system is desired to be composed by two applications – a mobile application for users to access all the implemented features, and a web application, for administrators to manage the overall system.

Given that the project's is being developed in conjunction with another author, the idea is to offer features related to training Programs, suchlike monitoring through individual performance feedback, customization, assistance, as well as recommendations based on personal characteristics. Even though the training Program aspect is exclusive to this dissertation, there are common components that were prosecuted with the other author, and some others that do not fall in this category but are required for the good functioning of the system.

The recommender system would then be able to manage and control the user's data, with the main goal of capturing their evolution and assisting them. It is desired, then, for such system to have an intelligent component, making recommendations through *AI* (*Artificial Intelligence*) techniques.

The registry of Program-related data should be user-friendly and intuitive, offering a personalized management of training Programs regimens to all types of individuals, allowing for them to consult training and evolution-related information, as well as access to that information by, possibly but not exclusively, a personal trainer or coach, empowering the user's results.

The main goals described will be achieved through the pursuing of the following specific goals:

- Investigation and analysis of the different state-of-the-art recommender techniques, to decide which one will be the most adequate for the current project.

- Investigation and analysis of different fitness applications to discover what is already being offered in the market and find what degree of differentiation is being offered.

- Development of a mobile application that allows users to perform Program-related tasks.

- Development of a recommender system capable of making intelligent recommendations.
- Development of an intelligent Personal Virtual Assistant (PVA) capable of communicating with the user in real time to answer their questions.
- Elaboration a study that evidences the utility of the development system followed by the results analysis.
- Fulfillment of the software development cycle.

1.4 Motivation

From a personal perspective, the motivation for this project rose from the necessity of having a featureful fitness app to use in the gym. Even though the market is fairly large and the offer vast, there was no specific one that catered to all the desired expectations.

The interest in the fitness and health industry, combined with the personal need of having a system with the most essential features and even innovative ones led to the selection of the present project which includes the creation of a system designed for people, who use the gym, with different goals, expectations and necessities, adapting itself to them in an automated way, providing numerous features for every type of user and assisting them in their activities, which will hopefully motivate them to continue pursuing fitness as a healthy life-style choice.

Some of the innovative features that were the motivation base for this project include the implementation of a recommender system, helping people achieving their goals through intelligent recommendation techniques, in an original way, since there's nothing of sorts well implemented in the market, as it will be discussed in further chapters.

1.5 Methodology

The current project was developed in conjunction with another author, which resulted in two different reports. Even though there are common components between both projects there's a clear separation at the domain level. That is, the present project is directed towards the "training programming" domain, and the other to the "exercises" one, each one with their respective features.

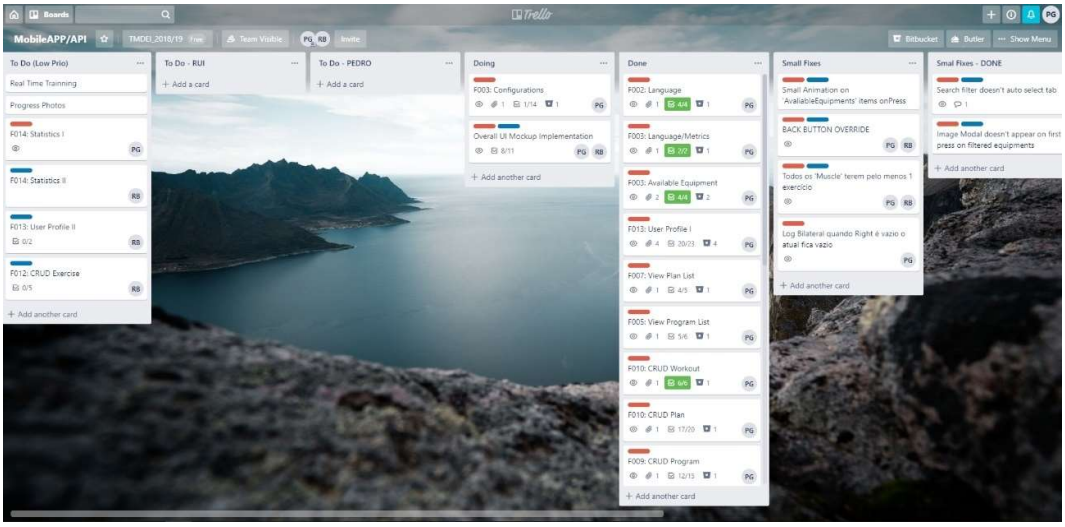

Figure 1 – Trello board

To ensure good collaboration and connection between both authors, some collaborative tools were used, including *Trello*, and *Bitbucket*, the first being illustrated in Figure 1.

1.6 Report Structure

In this section, the structure of the overall document will be exposed, briefly describing each chapter with relevant information, providing an introduction to the reader of the matters that will be discussed in each one.

The Value Analysis chapter, as the name suggests, will be dedicated to the presentation of the analysis relative to the value of the system. It's also in this chapter were the context of the project is presented at its entirety.

In the State-of-the-art chapter, the most developed piece of knowledge regarding recommender systems will be presented and discussed. Given that the project's distinctiveness and innovation in comparison with other similar projects comes from the introduction of a recommendation system, it becomes clear that the state-of-the-art should focus on these systems and in their impact, as well as debate which one is better for which context and why, and also dive deeper in recommender systems as a whole – exactly what is being presented in the this chapter.

In the Existing Solutions Analysis, the systems that have the most similarities with the one being developed will be addressed. After building and applying a selection criterion on a big group of systems, the resultant ones will be presented and analyzed thoroughly. At the end, based on the defined evaluation criterion, they will be compared and evaluated, attempting to conclude what could the new system bring to the market and how it could stand out.

As a little side note, the reason why the previous two chapters where not compiled in a single "State-of-the-art" chapter was due to several reasons: First, considering that for organization issues there are several levels of sub-sections, joining the two chapters would do more harm

than good by increasing even more the said levels, resulting in a poorer reading experience; Second, it makes the most sense to separate recommender systems and the systems discussed in the “Existing Solutions Analysis” chapter because they debate two very different concerns.

The Design chapter contains information relative to specific aspects of the system. In this chapter, the engineering of the crucial requirements will be presented, followed by the exposition of architecture-related information. It is in this chapter that the first decisions relative to the outcome of the system are made.

In the Implementation chapter what was effectively implemented in the system is documented, explaining the whole implementation process with the use of code snippets and other tools for better understanding. Also, it is also presented information regarding how the system managed to fulfill the projected non-functional requirements. Finally, a section with the tests that were conducted is also presented.

The Evaluation chapter, all the different crucial information necessary to the evaluation of the new system is defined and described. This information is composed by the definition of metrics, the formulation of hypotheses and the to-be applied methodologies. The results are then analyzed, through result and experimental analysis.

In the final chapter Conclusion, a summary of the overall project is presented through the enumeration of accomplished goals. The limitations and possible future work are also addressed, followed by a final appreciation.

Moreover, there’s also additional information attached in the end of the document. Information that is not crucial but complementary for the understanding of the project.

2 Value Analysis

This chapter will be dedicated to the value analysis of the system. According to the *Business Dictionary*, the value analysis "identifies and selects the best value alternatives for designs, materials, processes, and systems." [9] In the current context, the main goal of the value analysis is maximizing the value of the system, at the smallest possible cost.

First, the context of the project will be thoroughly presented, within its entire scope. Then, *Koen*'s innovation process – *New Concept Development Model* – will be presented, followed by the overall value of the system, the value proposition and the Canvas model.

2.1 Context

In this section, the context of the project will be presented, divided in the relevant distinct categories that make up the end product. Given that it's a mobile application directed towards fitness-related use, it seems only logical to put to context what exactly does fitness mean, why it is productive to explore this area and how is it connected with another ever-growing industry that is the mobile one.

Also, there is one important matter that needs to be addressed – even though this project is directed towards the development of a fitness app, this report will only contain the programming of trainings aspect of it (in broad terms). Given that the project is being developed in conjunction with another author, concerns had to be separated and so, this report will contain the mentioned aspects, and the other author's will contain workout and exercise's.

In this way, there will also be a section dedicated to the contextualization of training programming and a final one with the exposition and definition of different relevant concepts.

2.1.1 Fitness

2.1.1.1 What is Fitness?

What is fitness? Well, that's a simple question with complex and rather diverse answers depending on who's being asked. A good rule of thumb is by starting at the definition from *Oxford Dictionaries* that states fitness is "[t]he condition of being physically fit and healthy." [10] With this definition, it can be presumed fitness is the ability to perform physical tasks whilst maintaining a healthy condition which can be both physical and mental.

It is now established what fitness means, but what does it mean to be fit? According to the *Academy of Nutrition and Dietetics' Complete Food and Nutrition Guide* [11], when you're fit you have: energy to do what's important and to be more productive; stamina and a positive outlook to handle mental challenges; reduced risk of many health problems such as heart disease, cancer, type 2 diabetes, and osteoporosis; the physical strength and endurance to protect themselves and a better chance for higher quality of life and a longer one too.

With this being said, being physically fit is important to handle physical challenges in the everyday life such as walking, playing with friends or training for the sake of improving one's fitness level, which simply means to be better fit to perform specific tasks. This can come in the form of sports or simply from a health improvement perspective. Examples of this kind of physical activity are playing football, going to the gym, competing in the Olympic games, etc. Obviously, being fit in these types of activities can also improve the fitness level of everyday life tasks (i.e. someone who competes in football won't have much trouble climbing some stairs). Even though the connection between them depends on the specificity of the task being performed, in a general point of view, according to an article published in the *Medicine & Science in Sports & Exercise* it is suggested that "cognitive skills trained in sport[s] may transfer to performance on everyday fast-paced multitasking abilities." [12]

The aforementioned type of physical activity (training for improving one's fitness level) is the one that is concerned in the context of this project, which will be further explored next.

2.1.1.2 Effects of being fit

As it was already stated, being fit brings major health benefits, and a great part of that comes from physical activity, but what are the implications of being physically fit? According to the *Centers for Disease Control and Prevention* (*CDC*) [13], only half of adults engage in the physical activity they need to help reduce and prevent chronic diseases, which may be tied up to fact that the same ratio of adults live with chronic diseases. Moreover, about $117 billion are spent annually in health care costs that are allegedly associated with inadequate physical activity. 1 in 10 premature deaths could be prevented by getting enough physical activity, which is the reason why *Dr. Ruth Petersen, Director of CDC's Division of Nutrition, Phyical Activity, and Obesity* states that "[i]f you could package physical activity into a pill, it would be the most effective drug on the market." [13]

It is clear and time-tested that physical activity is directly connected with improved health and numerous articles and studies corroborate this statement, which is why *Healthy People 2010* [14] indicates that physical activity is the leading health indicator on a given population. [15]

2.1.1.3 Evolution and growth of the Fitness industry

According to the *IHRSA*'s (*International Health, Racquet & Sportsclub Association*) 2017 global annual report [16], in 2016, the "global health club industry revenue totaled $83.1 billion" and in the 2018 global annual report [17], in 2017 the revenue peaked at $87.2 billion, almost 5% more than the previous year, with an increase from 162 to 174 million consumers worldwide, which estimates a near 7.5% increase in the same time interval.

In the UK, *Allegra*, a leading-edge research and strategy consulting firm based in London, with the scope of producing a report regarding *UK's Fitness Club Market*, estimates that the said market is at "£5.1 billion with [an] annual growth of 7.1%" [18] that will still be positive over the next 5 years. Also, in China, according to the *IBISWorld Gym, Health & Fitness Clubs* [19] [20], over the past five years (from 2013 to 2018) the annualized growth has been of 10.4% with a $7 billion revenue in total for the year of 2018. In the US, this growth has been smaller, at 2.6%, but with a staggering $33 billion total revenue for the year of 2018. The US is still considerably very much ahead, with 11.5 times more number of business than China (112 thousand and 9 thousand) and 3.2 times more employment in the industry (800 thousand and 246 thousand), but according to Theo Hendriks, CEO of Sports and Leisure Group, "China could become the biggest fitness market in the world within the next 20 years. If only 4 percent of Chinese people join gyms, the country will need to build 30,000 new clubs over the next two decades." [21]

In Portugal, according to *AGAP* (*Associação de Empresas de Ginásios e Academias em Portugal)* [22] in 2017 the total revenue reached 220€ million with more than half a million members (535 thousand) throughout the country. Portugal is still far from coming close to the top markets across the globe in terms of both total revenue and memberships, since the leader is the U.S. with a revenue of $30 billion, followed by Germany and the United Kingdom, both closing in on the $5.5 billion mark [17].

2.1.2 Fitness and Technology

Unquestionably, the fitness industry is on the rise, and that is due to many factors, but the most relevant to the context at hands is the conjugation with technological devices such as smartphones. In fact, according to a Forbes article [23], the introduction of many *wearables* is one of the reasons why the fitness industry is growing. According to the article, once people start paying attention to digital feedback relative to their health (i.e. blood pressure and heart rate), "they start making more healthy decisions" [23]. As also stated, "[the] trend toward incorporating health data into [people's] daily lives isn't going away anytime soon" [23].

Almost 68% of the world population possesses some kind of mobile presence, that is a fact that can't be overlooked when trying to produce something for user consumption. There is no question that mobile-related technology will continue to grow exponentially, and to prove this point, some statistics will be presented, all taken from *Statista*.

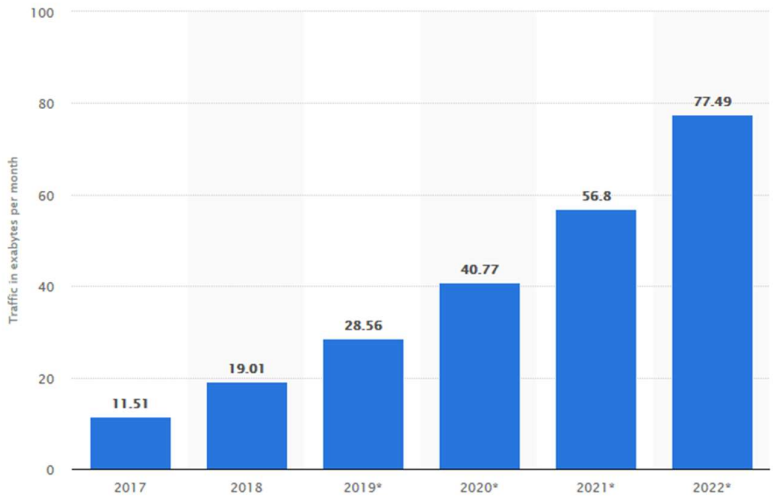

Figure 2 - Global mobile data traffic from 2017 to 2022 (in exabytes per month)[1]

In Figure 2, it is observable that projections indicate that between 2018 and 2022, the mobile data traffic will more than quadruple.

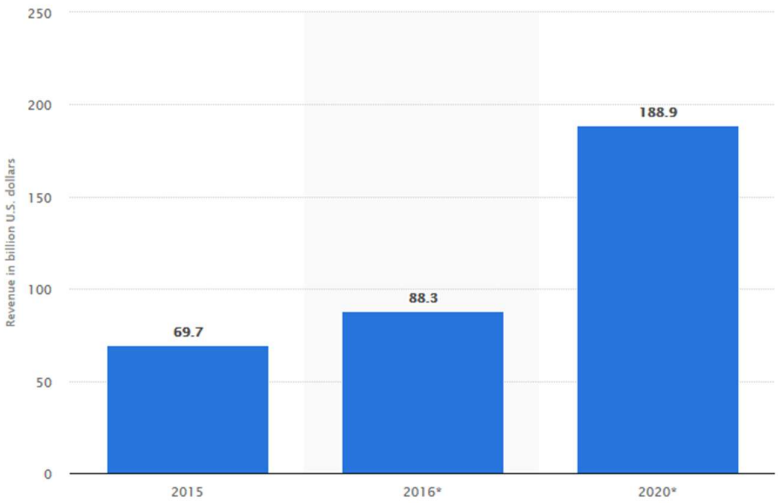

Figure 3 - Worldwide mobile app revenues in 2015, 2016 and 2020 (in billion U.S. dollars)[2]

In Figure 3, the projection indicates that between 2016 and 2020, the revenue from mobile apps will increase to more than double, reaching almost $189 billion.

There are countless more statistics and projections that can be found, and probably 100% of them indicate a growth in the mobile sector. A compilation of these can be found at https://clevertap.com/blog/mobile-growth-statistics/.

[1] Image from https://www.statista.com/statistics/271405/global-mobile-data-traffic-forecast/
[2] Image from https://www.statista.com/statistics/269025/worldwide-mobile-app-revenue-forecast/

According to *Flurry* Analytics [24] from 2014 to 2017, the health & fitness app usage grew by over a staggering 330%, which is something that can be overlooked. However, the growth is slowing down, which can be explained by subcategorizing health & fitness apps into four different categories: workout & weight loss; general health; nutrition; studios & fitness content. [24]

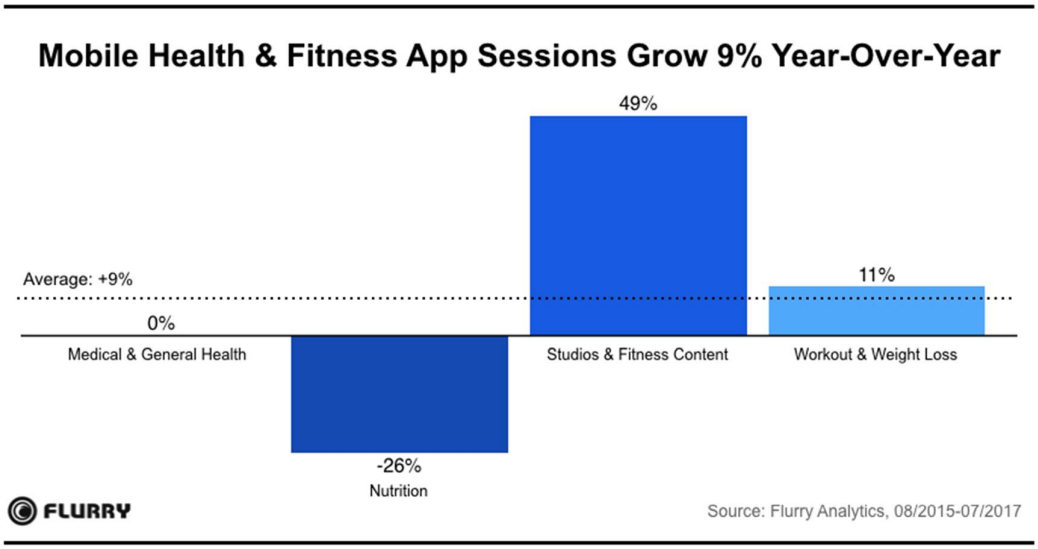

Figure 4 – Mobile Health & Fitness app sessions grow 9% year-over-year[3]

As Figure 4 depicts, even though general fitness content-like apps are still having a 49% year-over-year growth, that is held back by the slower growth of workout & weight loss apps, which have still a 11% growth and general health ones that stagnated, but more importantly by nutrition apps, that declined and are experiencing a decline of 26%. As stated by a *Flurry Analytics* report "[t]he negative trend of nutrition apps is likely influenced by studio & fitness content apps that are offering nutrition content in addition to their core focus." [24] This doesn't necessarily means that nutrition type apps are not needed, it simply means that more and more fitness content apps are incorporating nutritional content, and specific nutrition apps are becoming futile.

Moreover, workout & weight loss apps accounted for 73% of all health and fitness apps, followed by 21% on general health, as depicted in Figure 5. [24] This means that working out and tracking weight loss are key features for health & fitness app users, and as it was already discussed, a key driver for this growth is the rising of *wearables* which according to *Kantar Worldpanel*, as of December 2016, 15.6% of U.S. consumers owned a smartwatch of fitness band, which continue to outsell more advanced smartwatches. [25]

[3] Image from https://flurrymobile.tumblr.com/post/165079311062/health-fitness-app-users-are-going-the-distance

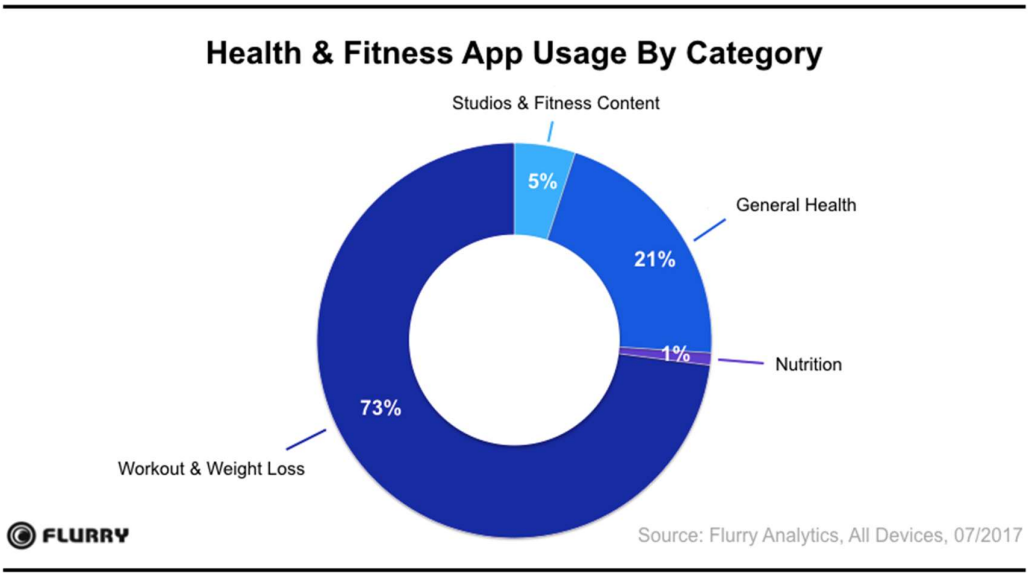

Figure 5 – Health & Fitness app usage by category[4]

In conclusion, fitness is a growing market, and so is the mobile one. Combining these two areas in a distinctive manner seems to be the logical course of action, through a mobile fitness app. Plus, according to *Flurry Analytics* "[h]ealth & fitness app users are the most loyal users in the app industry, with high retention rates, engagement, and frequency of usage" [24] and also, because all data indicates the continuous growth of *wearables*, seeking to penetrate the health & fitness app market seems like a great opportunity, now more than ever.

2.1.3 Training Programming

As previously established, the present project focuses on the training programming aspect of the system and as so, it's required to contextualize all the specific information regarding it.

Firstly, there's the need to define some key concepts:

- ***Reps:*** Number of repetitions a specific exercise is to be performed for.
- ***Sets:*** Series of reps of an exercise done sequentially (usually with rest between sets).
- ***Exercise:*** Combination of the number of sets and reps for a specific exercise)
- ***Workout:*** Combination of different exercises for a single repeatable session.
- ***Training Plan:*** Combination of different workouts for a single repeatable week.
- ***Training Program:*** Combination of different organized and customizable training plans, usually programming between 4 to 16 weeks. It can be seen as a series of repeatable

[4] Image from https://flurrymobile.tumblr.com/post/165079311062/health-fitness-app-users-are-going-the-distance

training plans, but the plans don't have necessarily to be same. The training program can be repeatable as well.

To better understand the described concepts, a full-scale training program will be built next.

WORKOUT A

Exercises	*Sets*	*Reps*
A	Set1	5
	Set2	4
B	Set1	12
	Set2	10

Figure 6 – Workout Definition

Figure 6 depicts the rough outline of a workout, composed by different exercises, each one with a list of sets and reps. Reading the workout, it tells that first, exercise A is to be executed, performing 5 repetitions on the first set, and 4 on the second. Following, there's exercise B, with the same logic – performing 12 repetitions of the first set and 10 on the second.

PLAN X

WORKOUT A

Exercises	*Sets*	*Reps*
A	Set1	5
	Set2	4
B	Set1	12
	Set2	10

WORKOUT B

Exercises	*Sets*	*Reps*
C	Set1	2
	Set2	3
D	Set1	4
	Set2	4

Figure 7 – Training Plan Definition

If two or more different workouts are combined, a training plan arises, as showcased in Figure 7. The plan is intended to be repeated usually weekly, so, in every week, workout A and B are performed (the days in which they are performed can be also specified).

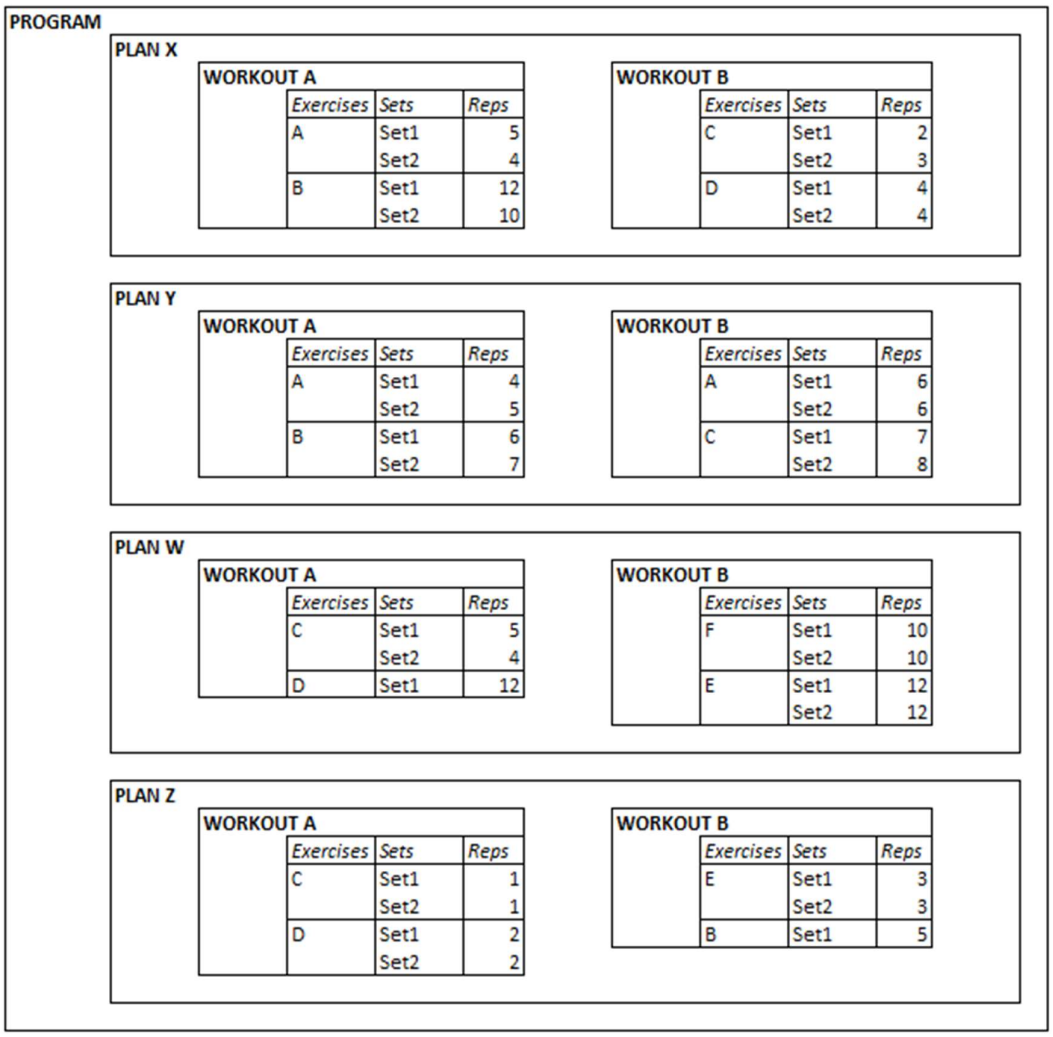

Figure 8 – Training Program Definition

Lastly, to build a full-scale training program, it's required to combine series of workouts (usually 4 to 16, but it might vary). As seen in Figure 8, the plans that compose the training program don't have to be necessarily the same.

2.1.4 Concepts

This section will be dedicated to the exposition and definition of some minor concepts that are, nevertheless, relevant to the project and need to be addressed.

2.1.4.1 RPE

RPE [26] stands for "Rated Perceived Exertion" and is a scale that ranges from one to ten that is used by athletes or coaches in order to "self-regulate their training intensity." [26] There are various ways to describe how the scale works, but a general rule of thumb is saying that an RPE

of 10 means that no more reps or weight could be performed/added. An RPE of 9 means that 1 more repetition could be performed with the same weight, and so on. There are also intermediate values like 8.5 or 9.5. The difference between 8.5 and 8 is the fact that even though with an RPE of 8, 2 more repetitions could be performed, with 8.5 the 2 repetitions are not a certainty.

2.1.4.2 1RM

1RM [27] stands for "1 Rep Max" and for some people is the ultimate end-goal. The 1RM is the most amount of weight an exercise can be performed for a single repetition. This value can be calculated by literally performing the exercise and testing it or by using a calculator that projects that value based on the most amount of weight lifted for more than 1 repetition. For example, using *Strength Level*'s calculator[5], if someone lifts 100kg for 5 repetitions, their projected 1RPM is around 112.5kg which means that, in theory, the same person could lift that weight for a single repetition.

Regardless, there are several calculators based on different and more parameters, in order to give a more accurate value.

Even though this is a projected, and therefore theoretical value, using 1RPM is extremely important specifically for strength straining. A study conducted for the *Journal of Sports Science & Medicine* concludes – "[…] a standardized 1RM testing protocol […] is a reliable measurement to assess muscle strength changes regardless of muscle group location or gender." [28]

2.1.4.3 Real-Time Training

For the purpose of this project, the concept of "real-time training" refers to the ability to follow a workout in real time. For example, being able to time the rest between sets, or to log information about the exercises as they're being performed is considered real-time training. An example of something that does not follow this definition, is logging information about a workout after it was performed and not during it.

Being able to end a set of an exercise, start the rest timer and log information about it during workouts is very helpful, because, among other things, allows users to log more coherent information. Obviously, describing something right after it was performed is much easier than after a long period of time where other similar things were also done.

2.2 New Concept Development Model

According to *Peter A. Koen* [29], the innovation process can be divided into three areas: the fuzzy front end (FFE), the new product development (NPD) process, and commercialization. FFE is usually considered one of the greatest opportunities for improvement of the overall innovation process. The fact that there's no common language and vocabulary to create new

[5] Data from https://strengthlevel.com/one-rep-max-calculator

knowledge and make distinctions between different parts of the process creates a shortcoming that was addressed with the development of a theoretical construct – the NCD model.

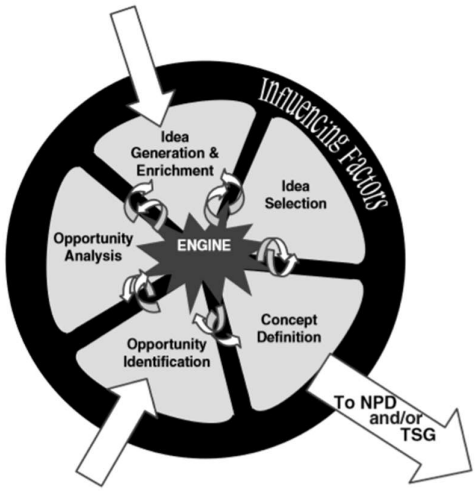

Figure 9 – NCD (New Concept development) model[6]

Figure 9 depicts the aforementioned model, and it consists in three key parts:

- The engine, which represents leadership, culture, and business strategy of the organization that drives the five key elements controllable by the corporation.
- The inner areas define the five activity elements of the FFE – opportunity, identification, opportunity analysis, idea generation and enrichment, idea selection, and concept definition.
- The influencing factors consist of organizational capabilities, the outside world, and the enabling sciences that may be involved. These factors affect the entire innovation process through to commercialization and are relatively uncontrollable by the corporation.

2.2.1 Opportunity Identification

It is in this section that the opportunities that might want to be pursued are identified. [29]

The opportunity for the project at hands rose from the perceived growth in the fitness industry. This is a trend that doesn't seem to stop anytime soon and capitalizing on the opportunity to provide a product or a service to a growing market is always a good thing.

Furthermore, another thing that strikes as an opportunity and also one of the reasons why the fitness industry is growing, is the introduction of many "wearables" like the Apple Watch and Fitbit, which is a huge opportunity for this project, that seeks to explore this trend.

[6] Image from "*Fuzzy Front End: Effective Methods, Tools, and Techniques*"

Moreover, as it will be analyzed in further chapters, the offer in terms of mobile applications specifically does not seem to satisfy the need at hands. However, the recognition of the different competitors will allow to pinpoint their weak aspects, in order to improve them, and the strong ones, to reproduce.

2.2.2 Opportunity Analysis

In this section, an opportunity is assessed to confirm that it is worth pursuing. [29]

As already exposed in the Context section, according to the *IHRSA*'s 2017 global annual report [16], in 2016, the "global health club industry revenue totaled $83.1 billion" and in the 2018 global annual report [17], in 2017 the revenue peaked at $87.2 billion, almost 5% more than the previous year, with an increase from 162 to 174 million consumers worldwide, which estimates a near 7.5% increase in the same time interval.

Additionally, as it was already showcased also, according to a *Forbes* article [23], once people start paying attention to digital feedback relative to their health "they start making more healthy decisions" [23], which is one of the reasons why the fitness industry is growing. Plus, as also stated, "[the] trend toward incorporating health data into [people's] daily lives isn't going away anytime soon" [23].

2.2.3 Ideas Generation and Development

It is this section that concerns the birth, development, and maturation of a concrete idea. [29]

In order to present various problem solving ideas, together with the different project interveners, a brainstorming was conducted and the collected ideas can be presented as such: implementation of a mobile/web app; implementation of a progress monitoring system; implementation of smart, AI based, techniques to generate different types of calculated recommendations; implementation of an in-depth profiling of users, to better adapt recommendations and others; implementation of machine learning techniques to improve the recommendations based on numerous variables; implementation of a personal virtual assistant, to help users in real time; synchronization with different wearables (i.e. smart watches) and other devices (i.e. scales); development of an image recognition system that detects movement and suggests improvements.

2.2.4 Ideas Selection

In most cases, the issue is not coming up with new ideas, but rather selecting the right ones to pursue in order to achieve the most business value [29], which is what this section is about.

From the identified ideas, the ones that fitted the solution better whilst simultaneously impose fewer limitations in terms of complexity were selected and can be summed as:

- Mobile/Webb app.
- Monitoring system.

- AI-based recommendations.
- Profiling.
- Personal Virtual Assistant.
- Synchronization with devices.

2.2.5 Concept Definition

This is the final element of the NCD and provides the only exit to the NPD or technology stage gate. In order to pass through, a compelling case for investment bust be made, described next. [29]

The project at hands has a clearly defined end concept, which is a mobile and web application with an individual user-tailored experience, automatization, recommendation and monitorization of fitness-related activities, more specifically gym ones.

2.3 Value, Perceived Value and Value to the Client

2.3.1 Value

There are two major types of clients/consumers of the product – gyms and regular gymgoers. The value brought to each is different and can be described as such: Gyms will have an added value to their business that can be translated in the retention of members, through the improvement on the direct connection between them and the staff, and through the offer of a superior monitoring and tracking system. Regular individual users will have all gym-related tasks such as information logging and progress tracking expedited, through the offer of the same system, unifying and computerizing all the specific individual needs.

2.3.2 Perceived Value

The perceived value will vary depending on the user, due to two reasons – opinion and profiling. First, obviously two distinct people can assign a different value to a same service/product, and second, the system is supposed to adapt itself to the individual needs of each user, so it's expected that the perceived value depends also on the profile defined.

Regardless, it's intended that users perceive value in the system based on the innovation it will bring in terms of the smart recommendation techniques, numerous integrations and unification of the most essential features, all in one intuitive and adaptive platform.

In terms of gyms, the perceived value comes in the form of member retention, based on the reasons described prior.

2.3.3 Value to the Client

The value to the client can be seen as the balance between the benefits they get and the sacrifices they need to concede in order to acquire such benefits. As so, it is expected that the project at hands creates a group of benefits that will counterbalance the sacrifices, that in this case come in the form of a paid subscription and ads. The benefits can be presented as follows: ease of use through an intuitive interface; innovation in multiple areas such as smart recommendations, personal virtual assistant and number of packed features; profiling that leads to a unique, customizable and adaptive experience; included support.

2.4 Value Proposition

The value proposition consists in the offering of a system based in a web/mobile application with the end goal of helping clients during their fitness journey. This will be achieved by providing numerous features such as: grouping of the most crucial functionalities; introduction of innovative features like AI-based recommendations, in-depth profiling and integrations; Expedite and improve the connection between a user and their PT/coach; Efficient progress monitoring; Reduced wasted time in planning and monitoring.

The presented features will be directed to any user, however, there is value added also to gyms themselves in the form of member retention, due to the offering of the said features to them.

Given that it is not the only system of sorts in the market, it's intended for it to emphasize on process automatization and computerization, accessibility of information and offering of a unique, individual and customizable experience.

2.5 Canvas

Essentially, the Canvas [30] model is a business model designed to allow all the business-related information to be presented on a single page, or in Figure 10 in this case.

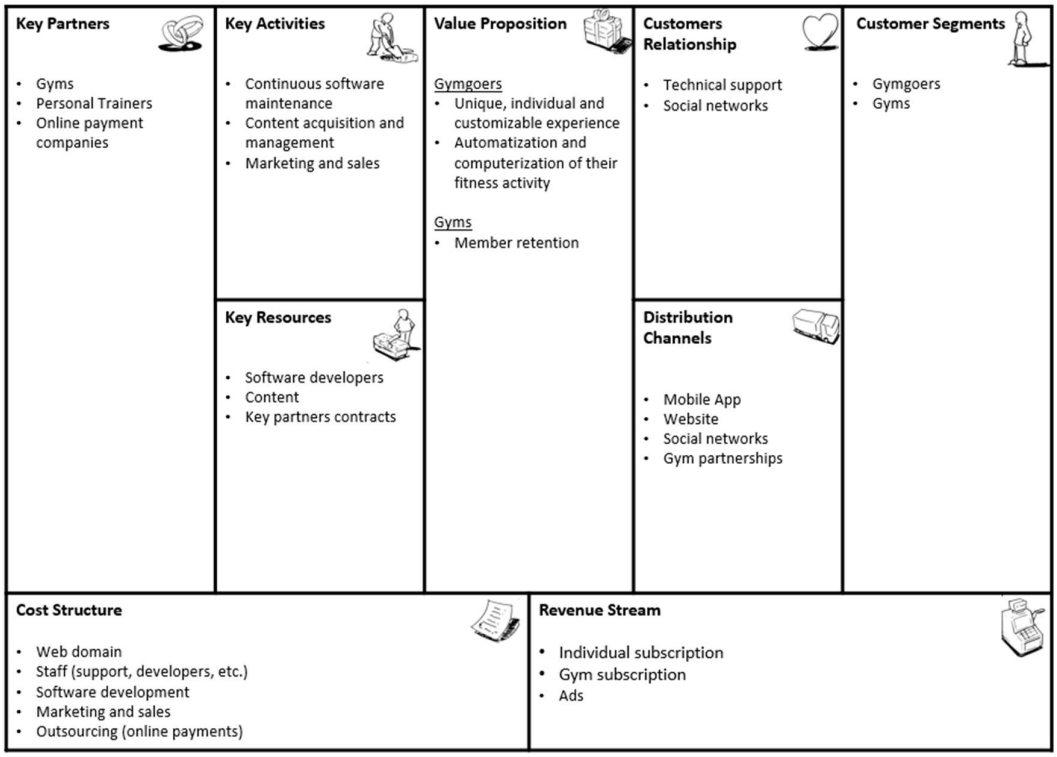

Figure 10 – Canvas Business Model

3 State-of-the-art

The state-of-the-art document is meant to evaluate the most developed piece of knowledge with regards to a specific topic, that in this case, refers to recommender systems. Considering that developing a recommender system is a major portion of the whole project it becomes evident that thoroughly researching the topic is of greatest importance.

According to the "*Recommender* Systems" [31], a driving force behind the development of these systems is the increasing importance of the web for electronic and business transactions. In this regard, the ease with which users are enabled to provide feedback about their preferences is an important catalyst. The most typical kind of feedback comes in the form of ratings, in which a user is prompted to associate numerical values relative to their likings on items – a notorious example of this is the five-star rating system. [31]

There are also other forms of feedback that are effortless to the consumer and even easier to be collect by businesses. An example of them is the simple act of purchasing or browsing items, which can be perceived as a manifestation of interest by the consumer. In this way, the main idea behind recommender systems is using the gathered data to deduce customers interests. That is because usually, past interests are often good indicators of future choices, and even though there are exceptions, most recommender systems are based on this relation between a *user* and an *item* - which refer to the entity to which the recommendation is being provided and the product (or service) being recommended, respectively. [31]

With this is mind, it becomes clear that the underlying principle behind recommender algorithms is that "significant dependencies exist between user- and item-centric activity." [31] This means that someone manifesting interest in, for example, a genre of books, is more prone to be interested in other books of the same genre rather than of different ones. Regardless, there may also exist correlations between various categories of items, which can be "learned" and applied to make more robust recommendations. As a general rule of thumb, the more data an algorithm has, the more coherent and robust will the predictions it makes be.

So far, the described recommendations can be seen as personalized recommendations, since they're molded to a single user's profile, which is built based on their past interactions with products (i.e. browsing or purchasing). But, there's also other types of recommendations – non-

personalized recommendations – which are much easier to implement, since they attempt to suit multiple users and not specific ones. These types of recommendations, even though effective in certain aspects, are not the focus of the study and will not be addressed any further. [32]

The following sections will unveil specific aspects of recommender systems including the different types there are and the issues and challenges associated with the employment of each one. Finally, in a conclusion section, the gathered information will be summarized and future prospects relative to the specific context of the project will be made.

3.1 Recommendation Techniques

In order for a recommender system to make a prediction, it first has to assess the usefulness of items relative to a certain user, to know exactly what is worth recommending. With this, the system must be able to determine an item's utility, or at least compare it to other items and base the prediction on the said comparison. This comparison is useful because recommendations "are offered as ranked list of items. In performing this ranking, recommender systems try to predict what the most suitable products or services are, based on the user's preferences […]." [32]

Considering that, there are multiple ways to make recommendations, in terms of the addressed domain, the knowledge and also the algorithm used – how the previously mentioned prediction of an item's utility is made. [32] The following sub-sections will be dedicated to the exposition of different types of recommendations and associated with each one, there will be some questions left unanswered, that will need to be addressed in further chapters, if the respective technique is chosen to be employed.

3.1.1 Collaborative Recommendation

This type of recommendation is based on *collaborative-filtering* (CF) which in its simplest and original implementation, recommends to a user what other users with similar profiles liked, or showed interest in the past. [33] For example, if two users have a purchase history that overlaps strongly and then one of them purchases a certain item that the other does not yet possess, then, the basic rationale is to propose it to that user, since it will mostly likely be of interest. "Because this selection of hopefully interesting [items] involves filtering the most promising ones from a large set and because the users implicitly collaborate with one another, this technique is called *collaborative filtering* (CF)." [34]

The standard method of application of the CF is known as the *Nearest Neighborhood* [35] algorithm which, simply put, aims "to calculate the similarities between [a] target user […] and all other users" with the goal to find the most similar ones, through different kind of processes.

The advantage of such technique is that the recommender system does not need to know specific data about an item (i.e. genre of a book and author), which means that it doesn't have to entered in the system. Regardless, using specific data to propose similar items ones might be more effective, since it's more robust.

Regardless, there are two different methods that are usually applied when dealing with this kind of filtering:

- ***User-Based Collaborative Filtering (UB-CF)***: Given a certain user x, the algorithm calculates the similarity of x with other users, based in the *Nearest Neighborhood* technique, finding the most similar ones, predict ratings for items x has not yet rated based on the ratings of similar users and recommend the top ones. [36] [37]

- ***Item-Based Collaborative Filtering (IB-CF)***: Similar to UB-CF, it calculates the similarity between any two items, predict ratings for the items the target user x has yet to rate, and recommend the top ones. [36] [37]

Anyhow, there are some questions that usually emerge when dealing with these types of approaches, and the most significant ones are as follows [34]:

- How to find users with similar profiles to the user for whom the recommendation is needed?

- How to measure similarity?

- How to deal with new users, with no defined profile yet (i.e. no history of purchases)?

- How to deal with new items, that no one has showed interest in yet?

3.1.2 Content-based Recommendation

One of the questions of the previously described recommender system - collaborative – is answered by this one. The question regarding what to do with new items that have yet to be purchased, for example, can be answered with content-based recommender systems. These systems base their recommendations on the characteristics of the items. If a user has showed positive interest in a certain genre of books, the algorithm is more likely to suggest books of the same genre. As described by "*Introduction to Recommender Systems*", "the similarity of items is calculated based on the features associated with the compared items." [32]

The advantage these systems bring in terms of new items, to answer the mentioned question, is due to the fact that the history of purchases (or any history associated with interests) is not needed, since when a new item is introduced, there can already exist others with similar characteristics, making it viable for recommendation without any history associated whatsoever.

There are also some questions that appear when working with content-based recommender systems [31] [32]:

- How can systems automatically acquire and continuously improve user profiles?

- How to determine similarity in terms of characteristics compatibility?

- What techniques can be used to automatically extract items characteristics?

- How to prevent “obvious” recommendations based on keywords that sometimes lead to the impossibility of an item being recommended due to the particular keywords it has?

- How to deal with new users, since a history of their interests is needed?

Even though *content-based* has some advantages comparing to collaborative such as the non-necessity to possess large user groups to achieve a reasonably accurate recommendation, and also the fact that it answers to the “new item problem” where an item couldn’t be recommended if it had no history associated, there are still some tradeoffs that come with it. Even though using the characteristics of items does seem like a more accurate way of identifying possible recommendations, acquiring such data automatically is an arduous process, “meaning that such information must be manually entered […] in a potentially expensive and error-prone process.” [34]

3.1.3 Knowledge-based Recommendation

In the specific context where purchases are not made very often, and there are many one-time buyers, *knowledge-based* recommendations are particularly useful. The fact that these kinds of purchases exist, means that the system cannot rely on history of purchases, which is a prerequisite for *collaborative* and *content-based* recommendations. Also, in these kinds of markets, such as the automotive one, the products evolve significantly over the years, making the preferences also show a corresponding evolution. Moreover, these products often have many different properties, and users might be interest in items with very specific ones which makes it hard to capture user interest. [31]

In such cases, *knowledge-based* recommender systems are very useful, where the ratings (or interest shown) in items are not used, but rather the “similarities between customer requirements and item descriptions, or the use of constraints specifying user requirements.” [31] An example of this are *constraint-based* systems, where explicit constraints regarding an item’s details such as color, model, price, etc. can be used to deduct interest. Moreover, these constraints may also be used to “describe the context in which certain features are relevant for the costumer” [31] such as, for example, that a car of smaller portions is advantageous if the customer is interested in parking in busy cities. As explained by “*Recommender Systems: An Introduction*”, applying explicit constraints to deduce relevant features is of great value. “Simply presenting products that fulfil a given set of requested features is not enough, as the aspect of personalization is missing, and every user (with the same set of requested features) will get the same set of recommendations.” [31]

Other relevant aspect of these types of systems is the “user interaction”. There’s a need to extract specific information about a user’s interest, and a fair approach would be to directly ask the user about their requirements. However, such an approach, “not only requires detailed technical understanding of the item’s features but also generates additional cognitive load in scenarios with large number of item features.” [31] More elaborate approaches try to incrementally ascertain preferences through an interactive and personalized dialog between the system and the customer.

There are some questions that beg to be addressed when dealing with *knowledge-based* recommender systems [31]:

- How to rank items based on the user's characteristics?
- How to acquire the user profile in a context in which no purchase history is available?
- How to take customer's explicit preferences into account?
- Which interaction patterns can be used in interactive approaches?
- How can the dialog be personalized to maximize precision of the preferences?

3.1.4 Hybrid-approach Recommendation

As the name suggests, taking a *hybrid* approach to recommender systems simply means to combine different techniques, like the ones mentioned previously. By doing so, one can "fix" the disadvantages of a certain system with the advantages of another. [34]

When combining different types of recommendations to adopt a *hybrid* approach, there are some questions that have to be answered [31]:

- Which techniques can be combined and what are, if any, the prerequisites to do so?
- Should proposals be calculated for two or more systems, or do other hybridization designs exist?
- How should the results of different techniques be evaluated, and can they be determined dynamically?

3.1.5 Matrix Factorization

One of the main problems with commercial recommender techniques, especially *collaborative-filtering*, refers to sparsity and scalability. In other words, these recommenders do not deal very well with the lack of data nor with the increased growth of users and items. As reported in a "*Knowledge and Information Systems"* article named "*Scalability and sparsity issues in recommender datasets: a survey*", even though nearest neighbor computation constitutes a typical approach for CF techniques, due to its high accuracy, "its performance on scalability is still poor given a huge user and item base and availability of only few ratings (i.e. data sparsity) […]" [35]

There are numerous and very diverse proposed approaches to alleviate the data sparsity problem, including a *Multidimensional* model [36], *demographic filtering* [37], *content-boosted* CF [38], and the most notorious and the one that will be explored in the context of this project, *Singular Value Decomposition* (SVD) [39].

SVD is the main technique used in an advanced method of recommendation, that "decompose[s] the original [user-item] sparse matrix to low-dimensional matrices with latent factors/features and less sparsity." [40] This advanced method is called *Matrix Factorization*.

Representing users' preferences with low-dimensional matrices offers great advantages in terms of *latent features*. In other words, if a user gave high ratings to a set of History movies, then there's a high probability that the same user might enjoy other History movies in the future. *Latent features* are expressed by higher-level attributes, which in this case refers to the History genre. As it was excellently put by an article on "*Towards Data Science*", "what matrix factorization eventually gives us, is how much a user is aligned with a set of *latent features*, and how much an [item] fits into this set of latent features." [40] This offers an advantage over other methods because even though two users haven't rated the same items, there's still a way to calculate the similarity between them by finding underlying tastes, expressed by *latent features*. [40]

Returning to SVD, since it's the main technique used, it's the most crucial piece of knowledge required to understand the workings of. Based on *Linear Algebra*, the theoretical background according to *Golub* and *Reinsch* [41], states that:

Let A be a real *m* x *n* matrix with *m* ≥ *n*. It is well known (cf. [98]) that

$$A = U\sum V^T \tag{1}$$

where

$$U^T U = V^T V = VV^T = I_n \ and \sum = diag(\sigma_1, \dots, \sigma_n)$$

This means that any matrix A can be decomposed into 3 matrices, U, $\sum$ and V. With an initial matrix A with r rows, c columns and rank m, where the columns of U and V are orthonormal vectors defining the *left* and *right* singular vectors of A, and $\sum$ is a diagonal matrix containing corresponding singular values "representing how important a specific feature is to predict user preference." [40] It can be said that A is a $m \times n$ ratings matrix, U is a user-latent feature $m \times c$ matrix, and V is an item-latent feature $r \times m$ matrix, and the singular values correspond to columns and rows of the original matrix respectively. [39]

For the sake of simplicity, the $\sum$ matrix will be removed from the equation [42] [43], since it simply acts as a scaler. Hence, it can be assumed that it was already merged into one of the other two matrices. Then, the final equation becomes:

$$A = UV^T \tag{2}$$

What this equation (2) ultimately means is that we can have a user-item matrix, A, and decompose it into two matrices that when calculated their dot product results in the original matrix. Why is this useful? Because of the simple fact that matrix A won't be nowhere to completely filled with data (or we wouldn't need recommendations), and by decomposing it, applying some math to it and recomposing it, we can obtain the original matrix with the previously empty spaces filled with predictions of ratings, as depicted in Figure 11.

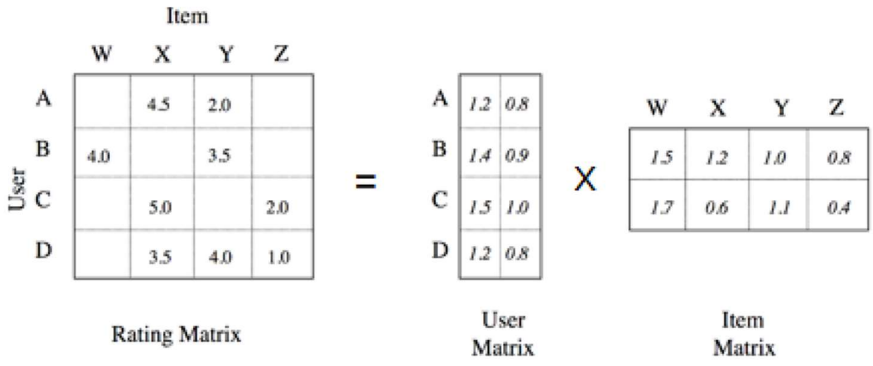

Figure 11 – Matrix Factorization example[7]

The aforementioned "applying some math" refers to the factorization of individual values. The rating of a user u for the item i, that will be denoted as r_{ui}:

$$\begin{pmatrix} & r_{ui} & \end{pmatrix} = \begin{pmatrix} & — \; p_u \; — & \end{pmatrix} \begin{pmatrix} & | & \\ & q_i & \\ & | & \end{pmatrix}$$

Figure 12 – Factorization of a Single Rating[8]

The previous Figure 12 represents the dot product of two vectors, p_u which is a row of the U matrix specific to user u, and q_i which is a column of the V^T specific to item i. [43] This can be summed as:

$$r_{ui} = p_u \cdot q_i \tag{3}$$

where "·" stands for the dot product

Applying the previous formula (3), all the empty values from the original matrix can be filled, producing ratings that when analyzed and ordered, produce recommendations.

The only problem with this method is that we cannot use pure math form of matrix factorization to predict values because then we will learn to predict zeros for missing data when decomposing the matrix into the other two. Thus, there's the need to find a different way to obtain the correct matrices to obtain the missing data based only on the non-zero values. The final question then becomes: How does one obtain the correct values for the orthogonal U and V^T matrices? This has multiple answers and therefore multiple ways to achieve the same result. Regardless, these techniques are called optimizers and depending on the context, different

[7] Adapted from https://medium.com/@connectwithghosh/simple-matrix-factorization-example-on-the-movielens-dataset-using-pyspark-9b7e3f567536
[8] Image from http://nicolas-hug.com/blog/matrix_facto_2

ones can be employed which can be detailed and explored in further sections providing that *Matrix Factorization* is a chosen recommendations technique for the system.

3.2 Issues and Challenges

Regardless of its usefulness, recommender systems also have their issues and challenges that need to be surpassed, if a precise, error-free system is what it's desired. The next sub-sections will be dedicated to exploring some of the most relevant issues and challenges associated with recommender systems.

3.2.1 Cold Start Problem

As it was already discussed when presenting collaborative and content-based recommendation techniques, when a new user or item enters the system, there's always an issue to be dealt with, due to the fact that there's no history associated with any whatsoever. This obviously means that predicting the user's interest will be less accurate and rating the items troublesome because they can't be recommended to anyone. There are some ways to solve this issue, according to "*Collaborative Filtering Recommender Systems*" [35]:

- Having the user rate some items initially before using the service, to set a ground base.
- Displaying non-personalized recommendations until the user has rated enough.
- Asking the user to describe their taste in aggregate (i.e. "I like science fiction movies").
- Asking the user for demographic information.
- Using ratings of users with similar demographics as recommendations.

Moreover, there are also some domains where there may be many "sleepers" – items that are very good but still unrated – and several techniques to recommend them include [35]:

- Recommending items using non-CF techniques, such as content analysis or metadata.
- Randomly recommending items with few or no ratings and asking users to rate them.

3.2.2 Synonymy

According to "*A survey of Collaborative Filtering Techniques*" [36] definition, a synonym "refers to the tendency of a number of the same or very similar items to have different name or entries." [36] For instance, an item labeled as "*horror movie*" and "*horror film*" are actually the same item, but memory-based CF systems would find no match between them. The degree of variability in descriptive term usage is greater than commonly suspected, which decreases the performance of CF systems. [36]

Some attempts to solve this problem were made, with "*Singular Value Decomposition"* techniques, particularly using the "*Latent Semantic Indexing*". This method is capable of dealing

with the synonym problem to some extent, but it only gives a partial solution to the *polysemy* problem, which refers to the fact that most words have more than one distinct meaning. [36] [37]

3.2.3 *Gray Sheep*

According to "*Combining Content-Based and Collaborative Filters in an Online Newspaper*" definition, a "*gray sheep*" refers to "individuals [in a community of users] who would not benefit from pure CF systems because their opinions do not consistently agree or disagree with any group of people." [38] Even after the first startup phase, these individuals will rarely, if ever, receive accurate recommendations based on CF.

A hybrid approach was offered by the aforementioned reference, in which CF recommendations were combined with content-based ones, basing the prediction on the weighted average between the predictions of both. "Moreover, the weights [of both predictions] are determined on a per-user basis, allowing the system to determine the optimum mix of content-based and collaborative recommendation for each user, helping to solve the *gray sheep* problem." [38]

3.2.4 *Shilling Attacks*

"Shilling Attacks" happen when someone give tons of positive reviews for their own product and/or negative ones for their competitors. It is, obviously, desirable to discourage this type of behavior through the introductions of precaution measures in CF systems. [36]

The effectiveness of these kinds of attacks has been already studied, and "*Shilling Recommender Systems for Fun and Profit*" found out that item-based CF algorithms were much less affected by the attacks than a user-based one. It is also suggested that new ways need to be used to evaluate and detect *shilling attacks* on recommender systems. [39]

There were numerous attempts to solve the issue with *shilling attacks*. In "*Effective Attack Models for Shilling Item-Based Collaborative Filtering Systems*", a partial solution to the bias injection problem was given, through the use of hybrid and model-based collaborative filtering systems. [40] Also, in "*Collaborative recommendation: A robustness analysis*" a contribute was made to solve the attack problem by analyzing robustness, a recommender system's resilience to potentially malicious perturbations in the user-item rating matrix. [41]

Moreover, Bell and Koren, in "*Improved Neighborhood-based Collaborative Filtering*" [42] used a comprehensive approach to the attacks, by removing global effects in the data normalization stage, and working with residual of global effects to select neighbors. [36] In 2009, they were awarded by *Netflix* on their *Netflix Prize*[9] initiative in which they substantially improved *Netflix*'s prediction accuracy with their algorithm - *BellKor's Pragmatic Chaos*[10].

[9] https://www.netflixprize.com/

[10] https://www.netflixprize.com/assets/GrandPrize2009_BPC_BellKor.pdf

3.2.5 Privacy

To have better recommender systems, with improved accuracy, feeding personal information is usually the best choice, but that may lead to privacy and security issues. Consequently, recommender systems should build trust amongst their users, however, CF recommenders are more prone to those types of issues. [36]

In CF systems, the user data is stored in a centralized repository, which can be compromised, resulting in data misuse. [43] Some techniques used to solve this issue include using randomized perturbation techniques, as proposed in "*Privacy-Preserving Collaborative Filtering*" [44], which allows users to publish their private data without exposing their identities, and also using Semantic Web technologies in combination with Neuro Linguistic Programming [11] (NLP) techniques, proposed in "*Ontology-Enabled Access Control and Privacy Recommendations*" [45] to mitigate the unwanted exposure of information. [36]

3.2.6 Sparsity

With the increase of users and items, the respective user-item matrix used to make predictions with collaborative filtering will become extremely sparse, leading to ever so less accurate recommendations. [36] [43] CF algorithms use the nearest neighbors' approach, and with the lack of data, computing neighbors becomes a very hard and challenging task.

This challenge can appear in different situations such as the already described *cold start* problem, where there's not enough information to compute neighbors of new users. This is especially difficult because new items cannot be recommended until some users rate them, and new users cannot be given good recommendations due to their lack of rating history. [36]

3.2.7 Scalability

The rate growth of algorithms used by typical recommender systems show a linear relation with the number of items and users, which makes it difficult for them to process such large-scale data. [43] To better handle this magnitude of data, several techniques have been proposed, including clustering, that searches in small clusters instead of the entire database [36], reducing dimensionality of data using *Singular Value Decomposition* (SVD) [46], and Bayesian Network. [47]

3.3 Conclusion

Having presented several techniques used to implement a recommender system and discussing the issues and challenges that emerge with them, it's now essential to draw conclusions from the gathered information.

There are numerous techniques that can be used to implement recommendations in a system and each one has their strengths and weaknesses. The most notorious ones were the ones

[11] https://www.nlp-techniques.org/

described in the previous sections and will be the ones that will most likely be selected when deciding a technique for the project. Choosing a technique to employ when trying to incorporate a recommender system will depend on multiple factors related to the context in which the system is intended to be integrated in. For this reason, in further chapters, a more thorough analysis will have to be conducted, with the purpose of gathering specific requirements for the recommender system, which will be the base of decision for choosing a recommendation technique.

4 Existing Solutions Analysis

The purpose of this chapter is to analyze modern solutions to the problem at hand – building a fitness app – with the intent of gathering information about what is currently being implemented and debating on what still needs to improve. To do so, four major types of fitness apps were considered, with the thought process behind their creation described thoroughly, and for each one two apps that fall into that category were selected. The selection and evaluation of the apps were based on rigorous criteria, described in the following sections.

Furthermore, the analysis of each app followed a set of common topics – the first one is just an introduction to the app, based on its own self-description, followed by its free and premium main features which, this time, are based on a direct usage of the apps by the author. Then, the strengths and weaknesses of the app are discussed through the eyes of the author, in what constitutes strong characteristics to replicate, and identify non-contemplated features from the existent solutions. Finally, for the applicable ones, the price plan is presented, and then some final thoughts summing the analysis is bestowed.

After all the relevant apps are discussed, a final chapter will collect the most pertinent information and compare them between each other, and also between the envisioned system to develop, in order to further inspect where exactly is the value and betterment.

4.1 App Types Definition

There is an absurd amount of fitness apps already clogging the market, and there is also a great deal of differences between them, which makes an arduous process to select and evaluate them. Since it is desired to gather relevant and representative information about the fitness market, having relevant and representative apps should be the number one priority. To help with this, four major types of apps were defined, making it easier to categorize every app in either group and subsequently discuss them amongst their peers, producing representative conclusions within that context (type) and generalizing the information with the purpose of comparing with other app types.

By analyzing different apps, two categories almost instantly created themselves, which make up a great percentage of the market: apps designed for gyms and apps for quick athletic workouts. The first category is self-explanatory, it refers to apps that are designed, usually, for gym chains that allows them to have their own system, calibrated to their needs and context. The second one is the one that appears more often in any fitness-workout related search, which describes apps arranged for people looking for small, quick, predefined workouts, that require little to no interaction from the user.

Even though these categories compose a great deal of the market, there were still "outsiders" that needed categorization. In sight of this, two more not so obvious types were defined: general apps and specific, customizable and flexible apps. The first one completes the market's percentage almost fully (after the first two categories being considered), and it comprises apps that are designed for people seeking more general goals, like muscle or strength building, that wish to maintain some sort of record of their activity. The second category is the "last 1%" of the market, which contains super specific, flexible, and therefore complex apps. It aims to help people with very well-defined goals that embrace the complexity of recording their activity as a trade-off for specificity and flexibility. These apps will be henceforth be called *PowerBuilding* apps, for reasons described next.

Briefly, the four categories can be defined as the following app types:

- **Gym Only:** Apps designed for gym chains for their own specific needs.
- ***PowerBuilding*:** Apps designed for super specific, flexible and complex goals.
- **Athletic:** Apps designed for small, quick workouts.
- **General Lifting:** Apps designed for general lifters.

In the next sub-sections, these types will be described more thoroughly, for the sake of consolidating their definitions and purpose.

4.1.1 Gym Only

As previously mentioned, these type's apps are part of the set that are limited to a specific gym and are adapted to that context. These provide the benefit of having a better, in theory, connection between its user and their gym from the perspective of scheduling classes and physical assessments, communicating with the gym specialists, for example, Personal Trainers and Nutritionists, and other perks that can be granted to the members that use the gym app like discount codes for products distributed by the gym.

For the average gymgoer it is undeniable the advantages that using their gym app offers, even from a comfort standpoint, considering that they are not required to search for an app that caters to their needs. Not to mention that usually, in most cases, the app is free for members, which can also be a deciding factor.

4.1.2 PowerBuilding

The *PowerBuilding* term comes from the junction between Powerlifting [48] which is a strength-based sport in which the contestants have three attempts at maximal weight on three different lifts, similar to Olympic weightlifting [49], where there are only 2 different lifts, and Bodybuilding [50], which is a sport that consists in the development of one's musculature and symmetry through the use of resistance exercises. The two sports come together as one when one's goal is to develop specific strength on the "three main lifts" and develop simultaneously a good physique. This is possible due to the fact that the resistance exercises are the same in both sports (in most cases), the only difference is in the way they're done,

The two sports included in the term *PowerBuilding* require the person practicing them to plan, program, and even log their activity. They're sports based in a great deal of variables that need to be kept in check [51] [52], like consistency, specificity, fatigue management, amongst others, which can only be achieved through intelligent programming and logging. This can be a highly complex and tiresome task to perform, creating a niche for that specific need.

As a result, this section was created with the intent of grouping apps that provide a more methodical and flexible approach to programming for bodybuilding and strength training. Even if some of them claim they're specifically designed for one of the sports, they're usually easily adaptable to the other, given their similarities.

As mentioned above, the great benefit of these apps is their methodical, flexible and customizable approach to programming. However, all this flexibility and customization comes with a cost – complexity. Even though most apps will claim they can pack advanced programming techniques and ease of use all in the same system, it's safe to assume that's not the case more often than not. Robustness comes with the cost of augmented complexity, and that's a cost that some are willing to concede. Some people, regardless of their end goal, want to maintain and follow a specific program, based in all kinds of variables like percentages, RPE, etc. and that's far from being completely straightforward, which is acceptable for them – they're not looking for easy, they're looking for results.

4.1.3 Athletic

By definition, an athletic person is someone who's "physically strong, fit, and active" [63] and that's exactly what the set of apps from the Athletic type are trying to portray they can offer – a means for anyone to be more athletic, in general, or towards a specific sport. The differences between these apps and the *Powerbuilding* ones, for example, range from the specificity of training to the exercise selection itself. Even though both can be adjusted for any needs, a Powerlifter would have a hard time using an athletic app for their programming, since yoga exercises and running are not exactly what they're looking for.

Nevertheless, athletic apps have their use and the majority of the population who works out will most likely adopt these apps to their training, since they provide a wide-range of exercises, stretches, running, etc. which, for most people, is what they're looking for anyway. Most people don't want to compete in Powerlifting or Bodybuilding, most people that go to the gym just want a healthier lifestyle, incorporating athletic exercises in their lives, and, for that, these apps are the best suited for them.

Even an athlete who competes in any kind of movement-based sport (e.g. Soccer, American Football, Basketball, etc.) will benefit greatly from using athletic apps, since, most times, they're looking to incorporate athletic movements in the gym to improve their sport specific performance.

4.1.4 General Lifting

If the goal is not to program and follow specific training regimens based on countless variables like it is on Powerlifting, nor to effectively increase the musculature development through intelligent hypertrophy programming as it is on Bodybuilding, and not even to improve sport-specific performance or overall athleticism, there's a chance the goal falls in the last category of fitness apps – General Lifting. General Lifting apps are designed to provide almost like the best of all worlds. They're not as specific as those of the other categories, but they provide more features and overall more flexibility.

People who use these apps, "general lifters", can be considered, generally speaking, as people that are not preparing themselves for anything specific like a sport, but instead are just looking to work out and better themselves and want to keep some kind of record of their activity and performance. It can be inferred, due to the lack of specificity on these apps, that these people are not very serious about lifting weights in the sense that they have no ambition past the goal of getting leaner, or stronger, or whatever their personal goal may be.

The difference between a "general lifter" and, for example, a Powerlifter, is that the Powerlifter has the goal of lifting the most amount of weight on specific exercises, in a specific manner, and wants to maximize the results through the use of intelligent programming and proper feedback, and the "general lifter" just wants to improve themselves and do what they like (in terms of exercises, for example). This does not go to say that one is better than the other in any way, people have different goals. Regardless, it's important to distinguish these types of lifters because even though they have similar interests in terms of documenting their activity (through the use of fitness apps), the way that activity is documented is very different and needs to be taken into account.

These apps can be easily confused for any of the previous ones due to their flexibility and wide range of features. As such, the biggest difference between them and the other apps is the lack of specificity. Even though you can program an athletic workout, you won't have as much specific features as you would have if you've used a specific athletic app.

4.2 App Selection Criterion

The apps, in order to be selected for analysis, have to meet certain criteria. This section will unveil the defined criteria as well as a justification for them.

The selection followed two different phases – firstly, many apps were gathered; secondly, the criteria was applied to them in order to diminish the total number of apps to a reasonable one.

The first phase consisted in the almost random selection of Health and Fitness apps from both the Google Play [12]and Apple store[13]. The apps were ordered by their rating, number of ratings, overall downloads, and other filters until a decent number of apps were gathered for each app type (Gym Only, *PowerBuilding*, etc.) with the exception of the Gym Only type, for reasons described below. With that, the fact that the apps were also selected to fit any of the app type can also be seen as a third criterion.

Then, in the second phase, the criteria of selection were outlined – the app had to have an average rating (between Google Play and App Store) above 4.5/5 with more than 30 thousand ratings in sum. Then, the criteria were applied to each app, and the ones that did not meet any of the criterion were discarded.

Table 1 contains the apps that resulted from applying the criteria "filter". All the listed apps have above a 4.5/5 rating with more than 30 thousand ratings through both store platforms. Some of the apps, like Nike Training Club even have a 4.7/5 rating with more than 369 thousand ratings, so it's safe to say that, in broad terms, the best apps were selected.

However, there is an app that doesn't meet any of the defined criterion and that is Intensity, with a 4.3/5 average rating and with only 271 ratings. The reason for adding this app to the selected ones that did met the criteria is because Intensity provides a unique aspect of specificity that none of the others did. Intensity fit the *PowerBuilding* type quite well, and given the fact that none other app did so, an exception was made.

Table 1 – App's Ratings[14]

Apps	**Google Play Rating (number of ratings)**	**Apple Store Rating (number of ratings**	**Average Rating (total number of ratings)**
Intensity	4.5 (244 ratings)	4.1 (27 ratings)	4.3 (271 ratings)
Strong	4.8 (~8300 ratings)	4.8 (~24200 ratings)	4.8 (~32500 ratings)
Nike Training Club	4.6 (~256600 ratings)	4.8 (~112600 ratings)	4.7 (~369200 ratings)
Freeletics	4.5 (~144500 ratings)	4.6 (~9800 ratings)	4.6 (~154300 ratings)
Jefit	4.5 (~63100 ratings)	4.8 (~12700 ratings)	4.7 (~75800 ratings)
Fitbod	Non-applicable	4.8 (~34400 ratings)	4.8 (~34400 ratings)

Finally, there's also two more apps that were not applied the criteria "filter" and those are the ones included in the Gym Only app type. These apps were specifically selected with one criterion only and that is – the ability to use them. Gym Only apps are exclusive to their

[12] https://play.google.com/store/apps/category/HEALTH_AND_FITNESS
[13] https://itunes.apple.com/us/genre/ios-health-fitness/id6013?mt=8
[14] Data from February 1st, 2019

respective members, so, being able to access them is quite hard. The only ones that were accessible were the ones selected.

4.3 App Evaluation Criterion

After selecting and analyzing the apps, there's a final section dedicated to their evaluation. That is important because even though the apps were individually scrutinized, there's always a need to sum the gathered information and present it properly in order to construct a conclusion.

To do so, the set of the most important functionalities were adopted to judge each app's capability of delivering the features that hold the most perceivable value all around. These features, as said, try to encompass all the features that are perceived to hold the most value in a desirable system, and will be applied to each app, to evaluate them and face them with the system that is meant to be designed. This will both assess the current offer in the market and the potential value the envisioned system could introduce to it.

The aforementioned features, used to evaluate the apps, can be described as such: profiling different users; create a single workout; start and/or register a workout; perform a workout in real time; log the workouts performed; workout statistics; info about each exercise; create a new exercise; create a training plan; create a full-sized training program; log personal body measurements; body measurements statistics; import progress photos and/or videos; directly communicate with a personal trainer or coach through the app; support multiple languages for the interface; have a social aspect (i.e. sharing workouts); useful calculators (i.e. calculators to quickly show how much weight should one put on each side of the bar); any type of integrations (i.e. Google Fit); ability to import and/or export information to the app; web/desktop application; any type of recommendations based on artificial intelligence techniques.

Also, it is necessary to evaluate the created app types. Each app type has a purpose, and it wouldn't make much sense to analyze them with no separation whatsoever. To do so, each app type will be evaluated by a set of specific features, usability and overall functionality. To sum the information, a table will be built with the strengths and weaknesses of each one, based on the criteria defined for them.

First, it is important to define the target "audience" for each app type, in order to better decide what would constitute value to them. That information is presented in the following Table 2.

Table 2 – App type targets

App Type	Target
Gym Only	Gym members.
PowerBuilding	Strength sport athletes, people seeking for strength training.
Athletic	Sports athletes, people seeking for an athletic lifestyle.
General Lifting	Average gym goers.

With that in mind, the criteria defined for each app type can be presented as such:

- **Gym Only**: Intuitive interface, given that most gym users would be beginners; ease of connection between the user and their respective personal coach and nutritionist; ability to schedule group classes; ability to schedule physical evaluations.

- ***PowerBuilding***: Program full-sized training programs; useful calculators; progress-oriented charts and statistics; multiple ways to program (i.e. percentages, RPE); well-implemented timers.

- **Athletic**: Fast usability, given that most users will be looking for a quick workout; progress-oriented charts and statistics; customization, to create different workouts/exercises.

- **General Lifting**: Versatility, given that it will be available for different kind of users; profiling of the different types of users in order to be more adaptive; multiple types of workouts, exercises, and others, to provide variety.

With the different evaluation criteria already defined and established, the next sections will be dedicated to the presentation of each app type and the analysis of each app, with the end goal being to sum the information and evaluate them accordingly. It is also important to mention that the evaluation criteria defined for each app type will serve as a guideline in the individual analysis of each app, more specifically in the strengths and weaknesses part. But, the said analysis, will not be restricted to those criteria and will be also based on specific characteristics each app might possess that are worth mentioning.

4.4 Gym Only

As it was already established, Gym Only apps offer distinctive features, but they also have their downsides. In the next sub-sections, some apps will be presented and discussed.

4.4.1 AmazinGym

AmazinGym [51] [52] [53] is a Portuguese gym app, designed for the gym members to have a platform in which they can perform multiple gym-related tasks directly from their device, as well as track workouts, log information and measure progress. It is certain that the ease of communication with the gym is the main reason why members use it.

4.4.1.1 Free Main Features

The fact that the app is meant to be used by the member and their respective personal trainer, the features vary from one another. The key features for the members are undoubtedly the gym related ones – ability to check the working hours of the gym and of all group classes and monitor daily physical activities and body measurements. Group classes, physical evaluations, and personal trainer meetings can be scheduled and cancelled.

Moreover, other features the members have access to are such as being able to check individual profile information (different logs, followers, etc.), monitor body measurement progress through statistics, as depicted on the leftmost screenshot of Figure 13, a calendar with the different logs of activities and sharing workouts. The member can also check their training plan with in-depth information regarding the exercises, the equipment needed, and others. Each exercise has a small video demonstrating how it should be performed, individual statistics for it, as depicted in the rightmost screenshot of Figure 13, muscles targeted and also a written description of it.

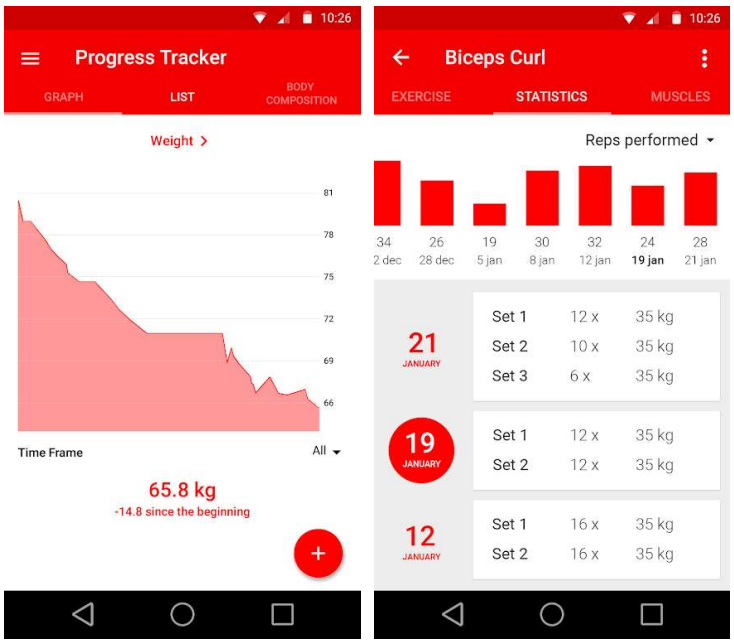

Figure 13 – AmazinGym's exercise statistics[15]

Additionally, the workouts can be started and performed in real time, logging the information about each exercise (i.e. number of repetitions) for posterior analysis of the personal trainer.

At the same time, the personal trainer is allowed to create and customize the training plans of each of its members and monitor their progress through statistics.

Finally, there's also multiple integrations with devices that allow to monitor the heart rate in real time, and with specific scales, to automatically import information relative to the member's weight, bodyfat percentage, among others, directly to the app. Plus, there are also some customizations that can be made, as far as units of measure and notifications go.

4.4.1.2 Premium Main Features

There are no premium features to mention, all of the features are free for members.

4.4.1.3 Strengths

With regards to the app's strengths, the ability to schedule group classes, physical evaluations and meetings with the personal trainer is definitely valuable. Plus, the available integrations with heart rate monitor devices and scales also adds huge value to the app.

[15] Images from https://play.google.com/store/apps/details?id=digifit.android.virtuagym.pro.amazingymmatosinhos&hl=pt_PT

4.4.1.4 Weaknesses

There are also some characteristics of the app that do strike as a weakness, and those are: non-intuitive interface, due to the fact of being arduous to find some of the features (i.e. scheduling group classes) and also due to the fact of the exaggerated number of different filters for exercises, which is confusing and sometimes not important at all; non-ability to schedule nutritionist consultations.

4.4.1.5 Price Plan

As mentioned previously, all of the features are free, and so, there is no price and subscription plan.

4.4.1.6 Final Notes

AmazinGym's app is undoubtedly an added value to their members, by offering features that bring them together, like scheduling classes or physical evaluations, but, the fact that some of those features are not easily accessible due to the non-intuitive interface is obviously problematic, especially for beginners.

4.4.2 Tripla Forma

Tripla Forma [54] Virtual Gym [55] is also a Portuguese gym app, designed for prescribing and monitoring training programs for Tripla Forma's members. It allows for the users to access their programs, prescribed by the respective personal trainers, and to register the training results for posterior analysis.

The direct communication with the personal trainer is the key aspect of the app and the reason the gym members use it.

4.4.2.1 Free Main Features

Considering that the app is designed to be used by the personal trainer, the nutritionist and the member in conjunction, the functionalities differ from one another. The features accessible by the member are pretty straight forward – one can consult their different workouts (there can be more than one), and also register results for each one. For each exercise, the member can send a message to their trainer, and can also see their progress in the form of a simple chart.

Furthermore, the member can also see the history of workouts, the goals established, in terms of nutrition, physical evaluation, etc. and, regarding the same physical evaluation, they can check the progress of the conducted evaluations, as depicted on Figure 14, where the different information like bodyweight, bodyfat, water percentage, etc. are presented and compared to another evaluation, measuring progress in the form of percentage.

Figure 14 – Tripla Forma Virtual Gym Physical Evaluation

Furthermore, the member can also consult their nutritional plan and customize certain aspects of the app such as the notifications and personal data.

On the other hand, from the personal trainer perspective, the features are the opposite of the members. The trainer can define and customize the different routines for each member they train, monitor their progress through their results and also answer the messages they send.

Finally, the nutritionist can define and customize each member nutritional plan, define goals and enter information regarding the physical evaluations performed by them.

4.4.2.2 Premium Main Features

There are no subscription plans, and so, no premium features to mention.

4.4.2.3 Strengths

Regarding the strengths of the app, there's only one thing that pops up, and that is the ability to have the data from bioimpedance scales [56], which measures the body composition, in bodyfat percentage, water, muscle, etc. on the app, and being able to compare it with previous evaluations, in order to measure progress. It can also generate graphs to better analyze the said progress, as depicted on Figure 15, where the graph for the bodyweight and bodyfat progress is displayed on the leftmost and rightmost screenshot respectively.

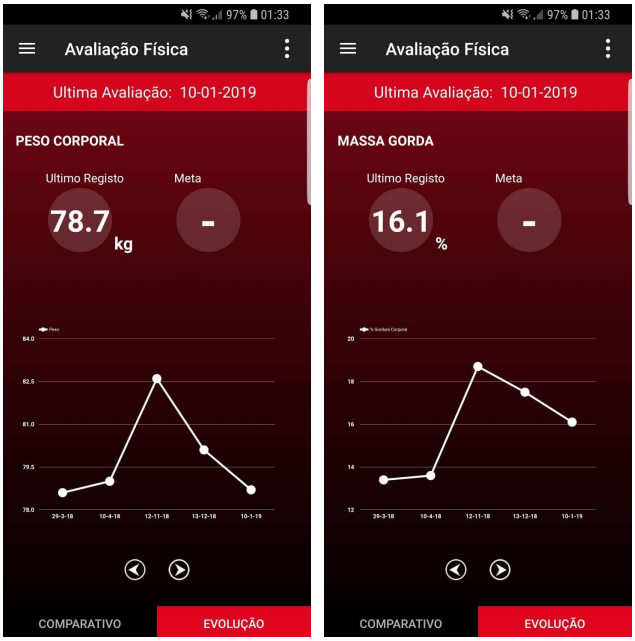

Figure 15 – Tripla Forma Virtual Gym Physical Evaluation progress graphs

4.4.2.4 Weaknesses

As far as weaknesses are concerned, the app is filled with them. For starters, the interface is not very aesthetically pleasing. Plus, the logging of information regarding exercises doesn't make much sense, since one can log the weight or the repetitions but not both. Also, there's no training program, only individual daily workouts that can be performed whenever the member sees fit. Given the fact the whole app revolves around workouts, one could assume the related features would be well implemented, but the fact that a member can log the same workout an infinite number of times on the same day tells otherwise.

4.4.2.5 Price Plan

As previously mentioned, there is no subscription plan to be mentioned.

4.4.2.6 Final Notes

There's no doubt Tripla Forma's Virtual Gym is useful for its members to monitor their physical evaluations progress, but besides that, the only ones that benefit from the features the app offers are beginners or people that are only looking to consult the random isolated workouts the personal trainers create for them. For people that do care about progressing in the gym, an app like this will take them no further than a sheet of paper with the list of exercises to perform.

4.5 *PowerBuilding*

This section will be dedicated to the presentation, analysis and discussion of apps that fall into the *PowerBuilding* category.

4.5.1 Intensity

Intensity [57] [58] [59] might very well take the number one spot when it comes to apps designed for strength straining. Specifically targeting Powerlifters, whose main goal is getting better on very specific movements, it's optimized for tracking progression with the use of their "progress-oriented interface" [57].

It's relevant to point out that Intensity also has a desktop version of the app, with the same features as the app.

4.5.1.1 Free Main Features

There are a number of features that do make this app shine from a free-user perspective, and one of them is the ability to access countless popular strength training programs already built for use. If that's not the intent, one can build day-to-day workouts or even customize the workouts of already built programs. However, one cannot customize or create full-scale programs, as that is a premium feature. It's also of great value to access in-depth statistics, in the form of graphs, about personal records, overall progression and others.

Other features, with substantial increased value are a timer and a stopwatch, a bodyweight tracker, numerous calculators and, to some extent, flexible customization, in terms of units of measurement (weight and distance), and not very much else.

Some features that are not required but are a "good-to-have" are such as messaging between friends and leaderboards with other users. These only add to the social aspect of the app, which is arguably necessary.

4.5.1.2 Premium Main Features

With this being said, there are also a big number of features that can only be unlocked via premium subscription. Only with it can one create their own custom program with their workouts and exercises. The free subscription only allows the customization of daily workouts, not the creation of an entire program. As it was already established, a workout consists in a set of exercises for a given non-repeatable day, and, a program is a set of planned workouts, usually outlining 4-16 weeks. In sum, if one's using a free subscription, they can only follow already built programs and customize daily workouts. If it's desired to customize an existing program, it's also required to possess a premium subscription.

Moreover, some other features that can only be accessed using a premium subscription are as follows: theming (changing themes within the app); upload videos of specific sets; sharing customized programs; generate programs automatically; priority support and feature requests.

4.5.1.3 Strengths

Intensity, no doubt, has numerous features, but the strengths that really highlight its potential range from the simplicity of the design to the intricacy of the statistics it offers.

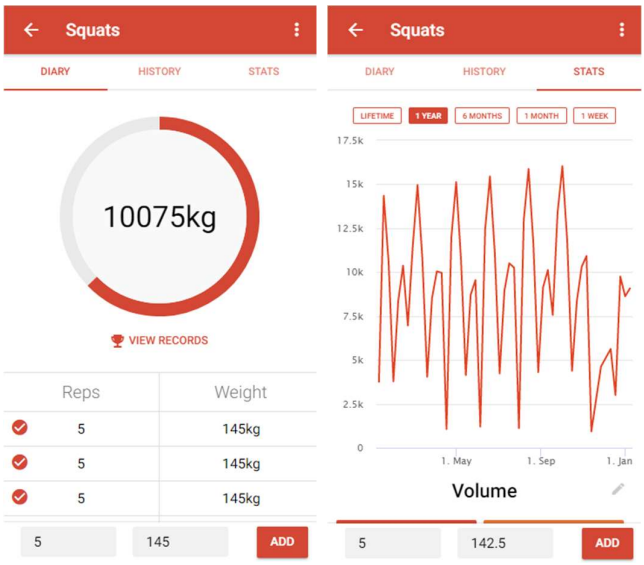

Figure 16 – Intensity's overview of a Set and Stats [16]

As observable in the Figure 16, the simplicity of the design is noticeable on both screenshots. On the rightmost screenshot, the said intricacy of stats is also visible, with statistics relative to strength progression, bodyweight progression, exercise and muscle group distribution, and others. Without going too much in-depth on very specific points, other strengths are the instant access to multiple popular programs and the ability to use them on the app instantly without any customization. The capacity to make numerous calculations within the app is also very useful, in order to calculate how much weight should be on both sides of the bar right away, or to time the rest interval, among others. Also, the relatively easy input of information is something that adds a lot.

4.5.1.4 Weaknesses

Because there's no good without the bad, Intensity also has some major weaknesses. By far, the greatest weakness is the need to have a premium subscription to access most of the essential features. With no ability to create custom programs or to customize existing ones, or to add videos to workouts, or even to change the theme of the app without a premium subscription, the use of this app becomes a fitting concern.

Regardless, looking at the app as a whole, some weaknesses that emerge are such as the inability to reset the current in-use program or even swap it with another one. When a program is selected, its workouts are automatically added to the correct day, regardless if there's an already planned workout for that day. If one selects a program and then selects another one, it

[16] Images from https://www.intensityapp.com/

overlaps the workouts with no way to completely remove one of the programs. Removing the workouts one by one seems like the only solution, which is a major weakness.

Moreover, even though the app provides a timer and a stopwatch, the concept of "real-time training" is nowhere to be found, since the two are completely disconnected from any workout set. To be blunt, using the mobile phone's timer and/or stopwatch serves the same purpose, which makes it a weakness, comparing to other apps that do have this concept well implemented.

Other weakness is the fact that it doesn't support multiple languages, only English, which makes it hard for non-fluent users, and also the inadequacy of good filter options. When searching for an exercise, one can only filter by its name and not by its muscle group, type of exercise, or others, which makes it really hard to build a decent workout without remembering the names of the exercises. Also, when adding a new exercise, for example, it's required to enter its muscle groups and exercise types but there are no pre-existent options, one has to type them in, making it impossible to group exercises by their attributes. All-around, the creation and filter mechanics are very defective.

4.5.1.5 Price Plan[17]

As covered above, most of the essential features are under the premium subscription plan, which raises a question regarding its price. The premium feature costs 4,39€ per month, but it also has a lifetime purchase of 17,99€.

4.5.1.6 Final Notes

In conclusion, viewing Intensity as a whole, it has to be said that it comes a bit short on expectations. Marketing itself as a system with great flexibility, numerous tools and "built for speed" [57], the usage of the app revealed to contradict almost every claim the brand formed. As said previously, the inability to create simple programs without having to pay for premium really makes the app nearly unusable without it, and even so, it's not possible to remove a program or swap it with another, or even filter exercises by any attribute. Even though it has a timer, there's no link between it and the exercises themselves and the statistics are not the most pertinent overall.

On another note, it's not all bad, as there are some ideas that can be taken from analyzing Intensity. The most important ones are the simplicity of the design whilst packing a somewhat featureful system and the relevant calculators, more specifically the plate calculator, which is very useful.

All in all, it would be a below average app without any paid features, but the fact that one has to pay to do most of the important stuff makes it not worth at all.

[17] Data from February 20th, 2019

4.5.2 Strong

Strong [60] [61] [62] was built to pack the most amount of features in the “simplest and most intuitive” [60] way. Designed to be adaptable to almost any workout and experience level, it’s safe to say that it will suit almost every need. Regardless of its simplicity and flexibility, it’s also very robust, as it provides tons of information about exercises and valuable insights regarding progress.

4.5.2.1 Free Main Features

There are tons of free features and the most important of them is the ability to start a workout very easily. The main screen of the app is a “Start workout” screen, that allows the user to start a routine from the ones already built or create custom ones with the desired exercises. It also allows the “quick start” of a workout, with the ability to add exercises in real time.

After starting a routine, whenever a set of an exercise is pressed, a timer automatically opens to count the rest time until the following set. This value can be customized in the exercise info, to specify a particular one, or to turn off the feature altogether.

Not only that, the exercises themselves have a few features associated such as a brief description and a small demonstrative video, a history of the sets performed and personal records with the number of repetitions executed, the max reps on a set, and others. The exercises can be easily filtered by their name, by the body part they work or their category and can be created as easily, providing the said filterable information. Additionally, there’s a quick filter option to select the most performed exercises.

When executing a workout there are other features like classifying the set, adding notes to the exercises, and also an indication of how much was previously lifted if the user wants to improve their record.

On another note, there’s a feature on the “Profile” screen that allows the user to see several stats related to their progress, like the workouts per week, and, if connected to Google Fit (or Apple Health in case of iOS), the calories per week, and daily macros.

Other small features consist on the changing the preferred units (weight, distance and size), a bodyweight measure tracker, a calendar with the history of past workouts and, for iOS users, the ability to associate a spoken word to a workout and then, using Siri, launch that routine verbally from outside the app.

4.5.2.2 Premium Main Features

Furthermore, Strong also has a premium subscription that offers a few more features suchlike the ability to store unlimited routines, charts with information about progress of specific exercises, a plate calculator that given a weight automatically calculates how much should be on both sides of the bar, such as the one depicted on Figure 17, a warm-up calculator that can be customizable to define how much warm-up sets should be performed and the percentage of weight they should be performed with. It also has a body measurements tracker, similar to

the bodyweight one but specific to individual body parts like the biceps, chest, etc. with charts displaying progression.

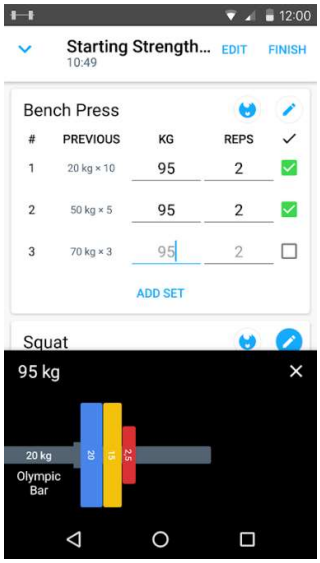

Figure 17 – Strong's Plate calculator

Apart from the presented premium features, there is one that substantially increases the value of the subscription, which is the integration with the Apple Watch through a fully-featured app that allows the user to log workouts and measure heart rate without having to access the phone, as depicted in Figure 18.

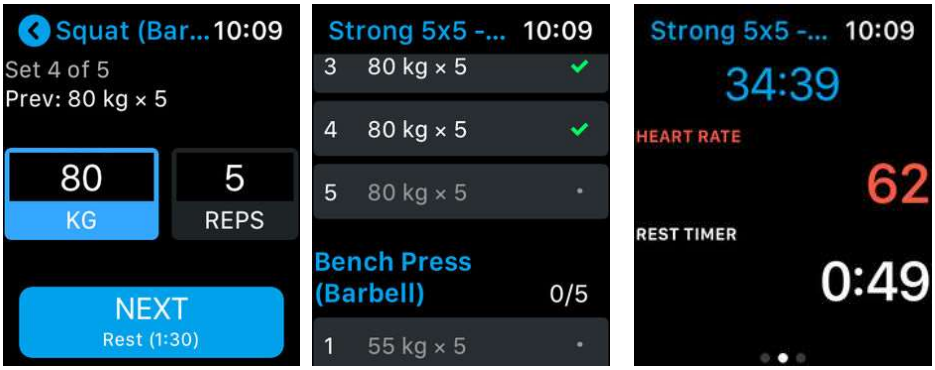

Figure 18 – Strong's Apple Watch integration[18]

4.5.2.3 Strengths

When it comes to strengths, Strong really does a good job with its workout-oriented system. There's virtually nothing that could be added to this methodology, with comprehensive descriptions of exercises, charts, records, real-time workouts, advanced plate calculators and

[18] Pictures from https://itunes.apple.com/app/apple-store/id464254577#?platform=appleWatch

numerous stats and insights relative to body measurements. The quick filter option for exercises really adds a lot to the interface, as it allows to immediately access the most performed exercises, also known as “favorites”. As depicted in Figure 19, the leftmost screenshot contains the list of all exercises, that can be filtered by 3 categories, as mentioned previously, but if desired, as shown on the rightmost one, the most performed exercises can be presented in order of usage.

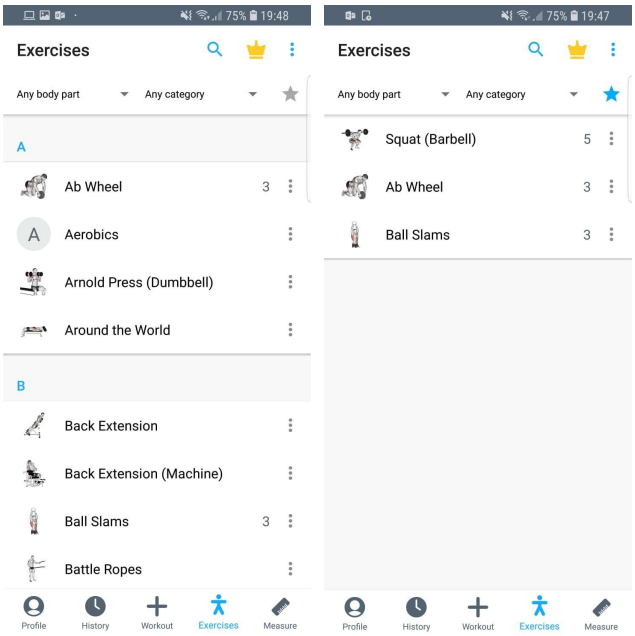

Figure 19 – Strong’s quick filter option for exercises

Even though the previously mentioned strengths are very useful, the one that really sets Strong apart is the integration with the Apple Watch. Notwithstanding that a premium subscription is needed, the fact that it’s possible to log the workouts through the Watch instead of the phone really puts Strong a step forward in usefulness. This integration is arguably the best feature in the entire app.

4.5.2.4 Weaknesses

On the other hand, there are also major weaknesses that might be determinant, especially for more experienced lifters, when it comes to choosing a fitness app. In all likelihood, a more experienced lifter will want to plan ahead a full-sized program, and that is simply not possible with Strong. Being a workout-oriented app, there’s no room for program planning, only for routine selection on the spot which is fairly unorganized. This is undoubtedly the most significant weakness since it questions the whole “strength training” aspect of the app. Additionally, there’s no way to relate sets with anything. An advanced lifter will want to base the set’s weight to a percentage, or even an RPE, but all that can be done is to establish a seemingly fixed weight.

Also, even though there are some already built routines, one can only store up to three custom ones with the free subscription, which is senseless because once again, for more experienced

lifters, there are countless different routines on a single training cycle (4-16 weeks), and the only way to save them is to either get a premium subscription, or delete and create routines every day.

Furthermore, there's also other weakening parts, such as the timers. Even though they do what they should, their only purpose is to count the rest time and overall workout duration. That data isn't analyzed anywhere, there are no statistics or graphs to transform the data into useful information. For example, it would be of value to know the progress of the rest time for a given exercise, or even the average duration of the workouts. Plus, the rest-timer is not very flexible or well-integrated. A rest-time is supposed to count the time between one set and the next, but Strong allows to perform all sets whilst still on the rest time of the first, due to the fact that there's no association between it and the individual sets. Plus, one can only set a default value for an entire exercise, not to specific sets, and even though that's not critical, it would be a good-to-have.

4.5.2.5 Price Plan[19]

The free version of Strong packs almost all the essential features and the premium subscription provides a complement to those. The price plan starts at 4.99€ per month, 30.99€ per year or 74.99€ as a onetime payment.

4.5.2.6 Final Notes

All things considered, Strong is a fairly decent app because its usable right away. One can immediately create custom routines (up to three), customize workouts and even create exercises which is more than enough for the average gym goer. The issue is that Strong markets itself even to experienced lifters, which is pretty absurd given the fact that it doesn't offer arguably the most important thing for them – the ability to set up a full-scale program. There's absolutely no credible powerlifter that would use an app that even paying 5€ a month wouldn't allow them to create a program. The closest thing to that would be to create all the different routines and try to remember on the day which one is supposed to be performed, and regardless of the fact that even for that a premium subscription is required, there would be absolutely no connection between routines, only a clutter of individual workouts, which is extremely chaotic. And only to add insult to injury, the sets have no relation with anything whatsoever. One can only define the weight and the reps of a set, which is sure more than enough for the average gym goer, but for an experienced lifter, there's a plethora of ways to program a set, based on RPE, percentages, and others.

To conclude, there's a lot of things that do make this app great for an average lifter, and even for an experience lifter some features are very useful such as the Apple Watch integration, but as a whole, there's unequivocally no way a decently experienced powerlifter would use it. Plus, for a serious lifter, the benefit that comes with a 75€ app is questionable at least.

[19] Data from February 20th, 2019

4.6 Athletic

As previously discussed, Athletic apps do seem like the most popular, but even though it may look like they pack the most advantages in terms of availability for the general public, since they do not tunnel vision on small niches (like apps designed solely for Powerlifting, for example), they still have their issues regarding overall specificity and flexibility, which will all be discussed in the next sub-sections, as the apps are presented.

4.6.1 Nike Training Club

Arguably the most popular fitness app, with over 10 million downloads on Google Play and being obviously very credible for the brand behind it, Nike Training Club [64] [65] [66] is the leading health and fitness app in the market.

With over 185 free workouts, ranging from boxing, endurance and mobility, to strength training and body-part focused workouts, there's no doubt this is an app of choice for sports athletes or general users seeking for a more athletic lifestyle. It also offers personalized workout recommendations, guidance and even workouts inspired on famous Nike athletes like Cristiano Ronaldo and Michal B. Jordan.

4.6.1.1 Free Main Features

Surprisingly, Nike Training Club doesn't have that many features, there's pretty much nothing besides performing a workout and adding other physical activities to the history.

On the first log of the app, a personal profile of the user is outlined, through the response of two questions regarding the sex and the frequency of physical activity per week. After that, the app recommends some plans adapted to the user's experience level.

The workouts are the main component of the app, as seen in Figure 20, the main screen is a list of top workouts selected based on the user's profile, new workouts, and others. There's also a possibility to browse all the routines by muscle groups, workout type, no-equipment workouts, etc. Plus, there's a tab, depicted on the rightmost screenshot of Figure 20, where one can access specialized guides with information about them, and the workouts to follow.

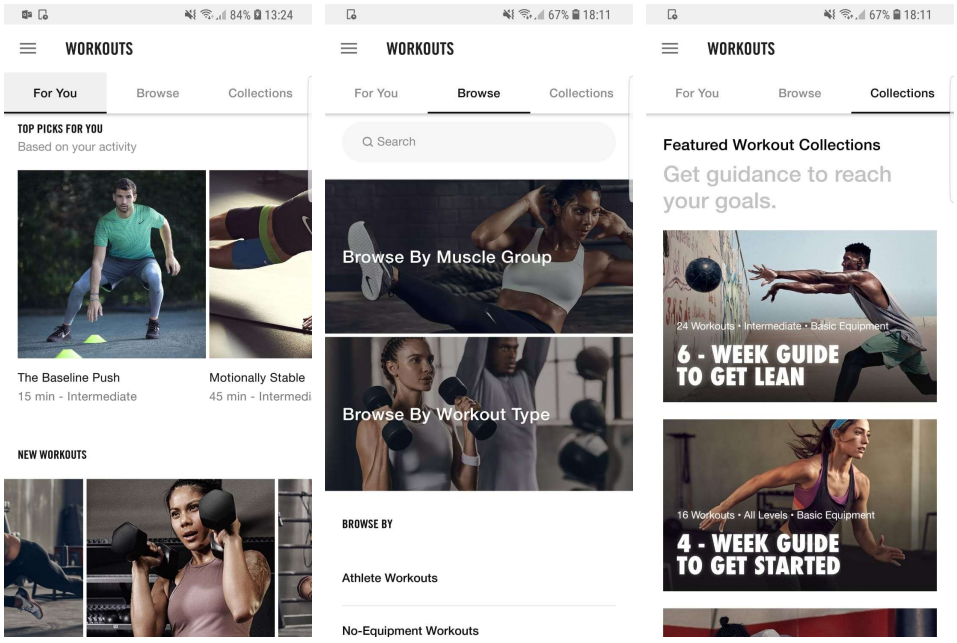

Figure 20 – Nike Training Club's workouts

A workout, more in depth, is characterized by its average duration, intensity and expertise level. To perform a workout, one has to first download it, then, after starting, it automatically shows a video of the first exercise with a timer, and, after that ends, the next exercise starts. The workout can be paused and finished at any time and, after completion, the user is asked how much effort was put in, and where the workout took place. It is also possible to follow a workout with an Apple Watch, removing the need to always be looking at the phone for instructions.

Furthermore, one can add a physical activity such as Running, playing Football or any other sports, Yoga, etc. The same questions as for the workouts are asked – effort and place. The Activity screen is composed by the history of activities and workouts performed.

Additional features can be summarized as: a news feed, an inbox for messages, a Nike event finder, some customization regarding the measurement units, and integrations with Google Fit, Apple Health and Nike Run Club, to automatically record all runs in the activity history.

4.6.1.2 Premium Main Features

All the features bestowed by Nike Training Club are free, as there is no premium subscription.

4.6.1.3 Strengths

There's no question the simplicity and overall look of Nike Training Club's interface is its major strength. It's very easy to start any workout and there are more than a few to choose from. The integration with the Apple Watch also makes the workout experience a bit easier, since it minimizes the interaction with the phone.

4.6.1.4 Weaknesses

Even though Nike Training Club is very solidly built, there are some weaknesses that can't be overlooked. The most essential one is the fact that there's no way to create workouts or even customize existing ones. There's absolutely no customization that can be made to the offered routines which makes it a bit unpleasant because one must follow workouts entirely built by other people and that's not very individual. Plus, as described previously, there's a section for "programs" that can reach up to 6 weeks. These can be mistaken by real programs but all they are is a few paragraphs describing the guide and then a bunch of workouts that the person has to choose from at random.

Additionally, another thing that falls short is the lack of progress-related statistics. All that is accessible is the history of workouts and activities, there's no real way to measure progress through indicators like the duration time, effort needed to complete, estimated burnt calories, heart rate variances, etc.

4.6.1.5 Price Plan

As previously mentioned, there's no premium subscription to be found, so, there's also no price plan.

4.6.1.6 Final Notes

In Summary, Nike Training Club is an app designed for people who're looking for quick athletic workouts and no commitment to a serious training regimen. Even though the app is very appealing, its content value is questionable, since it doesn't provide a platform for people to create their workouts, or to measure progress, it is merely a workout database.

Regardless, assuming that that's the intent, even the profiling made in the first log of the app is underwhelmingly feeble. There are tons of variables that can be used to profile different athletes, but picking from just sex and frequency of physical exercise is unfortunately lazy and generic.

Nike Training Club is an example of a superbly simple and good-looking interface, but it falls short on many levels. Given the size of the company behind it, it would be expected a bit more effort on questions like profiling, exercise customization and progress-oriented stats.

4.6.2 Freeletics

Freeletics [67] [68] [69] is a set of fitness apps, each with their own objective. The one being analyzed is the Bodyweight one, since it's the most popular and the most "Athletic" one. For the sake of simplicity, from now on, Freeletics Bodyweight will be addressed as only Freeletics.

Freeletics takes advantage of the "High Intensity Interval Training" workout methodology, or HIIT for short, which is essentially alternating between very intense anaerobic exercises and

short recovery periods. The app guides the user through these workouts, that range from 10 to 30 minutes.

Whether the goal is to "[l]ose weight, gain muscle or simply get in better shape" [68], Freeletics is the app to take.

4.6.2.1 Free Main Features

Freeletics is designed to be used as a premium subscription app, since its main feature is paid, but looking only at the free ones, one can assert that it is a workout database, just like Nike Training Club, the only difference being that it allows for a user to select and perform a single exercise, and not a full workout.

A workout can be selected from the workout list, where it can be filtered by its expertise level (beginner to advanced), duration (short to long), the body part it works, and others. Each one is composed by the equipment needed to perform it, explanatory videos, the summary of the included exercises and its duration and difficulty. When a workout is started, a timer begins, counting the duration of the first exercise. When the user finishes that exercise, they tap the timer and next one starts, and so on. On the end, it is required for the user to give feedback on the workout, providing information regarding the quality of the used technique. Similarly, one can also perform just one exercise, for a desired number of repetitions.

Furthermore, one can find training spots on the area and log runs, with the Freeletics Run app integration. The workouts and runs are logged to the user's profile and they can also be seen on the feed, where news and workouts from friends are posted. Besides that, and some small customizations regarding weight training units and personal information, there's not much more free features.

4.6.2.2 Premium Main Features

The thing that does really make this app shine is its virtual Coach [70], which is a "personalized training plan that uses a state-of-the-art artificial intelligence" [70]. The Coach adjusts itself to the user's fitness level, and based on the feedback given after each workout, it learns the individual strengths and weaknesses.

The way the individualization of the coach works is by knowing the user's personal information, recommending a Journey (which is a training program), like the one depicted on Figure 21, and then learning from the user's choices on number of training days, available equipment, limitations, among other. Based on the user's preferences, the Coach will suggest weekly training programs and at the end of those it will optimize it for the following week. At the end of the Journey, the user can change to another or try again the same one.

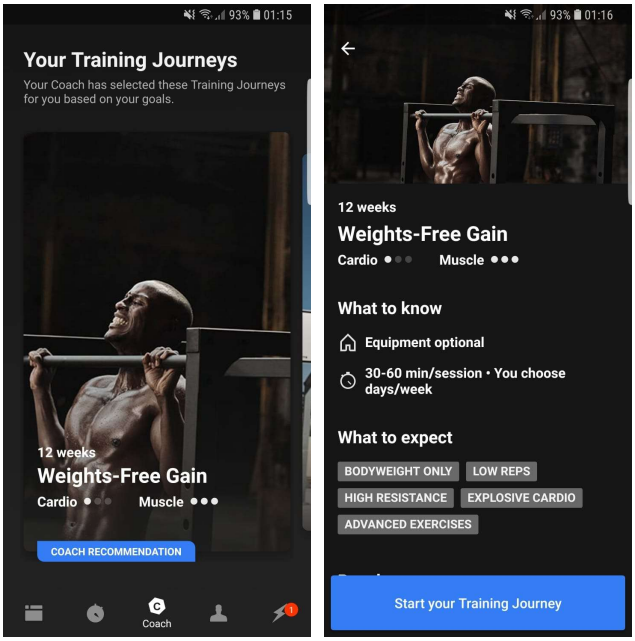

Figure 21 – Freeletics Training Journey

Even though the Coach is the main premium feature, and the reason that the premium subscription exists, there are also other perks from having it, suchlike accessing even more workouts, as seen on Figure 22, where some free exercises ("Krios") and also premium ones (i.e. "Aias" and "Elektra") are presented.

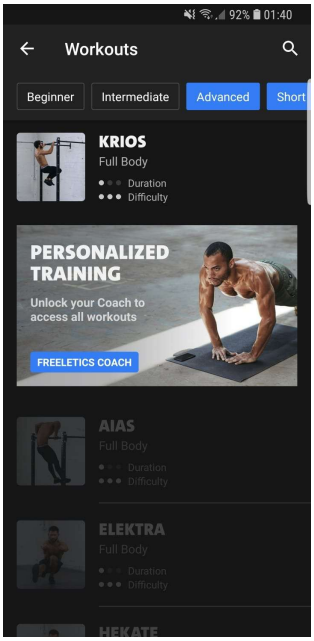

Figure 22 – Freeletics workouts

4.6.2.3 Strengths

There is no doubt the biggest strength of Freeletics is its virtual Coach, that makes the overall workout experience much more pleasant and individual. From a user perspective, being able to hand out the task of building custom individualized workouts to an automatic system is very helpful and valuable.

Also, the general appearance of the interface is quite appealing, and the usability is intuitive.

4.6.2.4 Weaknesses

Although the Coach offers a great deal of automatic customization, apart from that there's nothing to do besides selecting and performing a workout. There's no way to create a workout based on specific exercises or to monitor progression with relevant statistics. Once again, it is more like a workout database, where you have to pay to even access most of the exercises.

Without the Coach, there's not much value on the app, and even that has its flaws. The main issue is the progression system – it only involves adding repetitions to exercises, which ultimately is not that great because eventually there'll be an overwhelming number of them. Instead, it should progress to harder variations of the exercises when certain criteria are met, for example.

4.6.2.5 Price Plan[20]

The price plan for the Coach and the premium subscription varies depending on the frequency of payment and it can be synthesized as so:

- Annually: approximately 42€/per year
- Biannually: approximately 62€/per year
- Quarterly: approximately 70€/per year

4.6.2.6 Final Notes

All things considered, there's no doubt Freeletics, in order to be worth using, has to be with a premium subscription. The Coach is a definite differentiation from the competition. Regardless of its quality from a more technological point-of-view, the idea of individualization is very enticing, especially for beginners, sports athletes, or general users looking for a more unique athletic workout experience.

Lastly, something that can't be left unmentioned is the quality of the adaptiveness the Coach offers. As it was already said, the progression system revolves around only increasing the number of repetitions based on the performance of the user, and as far it is known, the Journey

[20] Data from February 20th, 2019

recommendations can be based on static features like the muscle groups, difficulty, duration, etc., and to do that, there's no need for "artificial intelligence".

Furthermore, the fact that one has to pay a minimum of 3 months to use the Coach without trying it first is a bit upsetting.

To conclude, even though the Freeletics app is very well designed, it lacks a great number of free features, and even for the Coach subscribers there are still some concerns to be pondered.

4.7 General Lifting

General lifting apps can be thought as the "jack of all traits" because they try to provide a range of different types of features. It is known that the expression doesn't end there and usually a "jack of all trades [is a] master of none", which means that they do not specialize in any of the features they pack. Furthermore, it is also known that a "jack of all trades, master of none, [is] often times better than master of one", which is what will be discussed in the next sub-sections, if the improved flexibility in these apps accounts for their lack of specificity.

4.7.1 Jefit

As far as having a big social community, with millions of members to share progress with, whilst maintaining a huge database of exercises and routines, Jefit [71] [72] [73] undoubtedly differentiates itself from the competition. Reportedly having more than 1300 exercises and 1100 HD training demonstrative videos, it would be very unlikely for someone to not find what they were looking for.

Even though it was an enormous database, Jefit is more than that, since it also allows to track workouts, create custom routines, consult body and lifting progress through the use of statistics and connect with other users, in order to share and compare progress.

4.7.1.1 Free Main Features

With regards to free features, Jefit offers a great number of them. The main part of the app is the workout/exercise section, where a user can view the workout and exercise list, select a workout as the active one, create a workout, filter exercises, among others.

Probably the most relevant feature is the ability to create a workout with any of the exercises and define characteristics such as the number of days per week it will have, its difficulty level, among others. Then, for each of the days, specify the exercises to perform, and also the number of repetitions, weight used, etc. It is possible to perform an individual exercise outside the workouts.

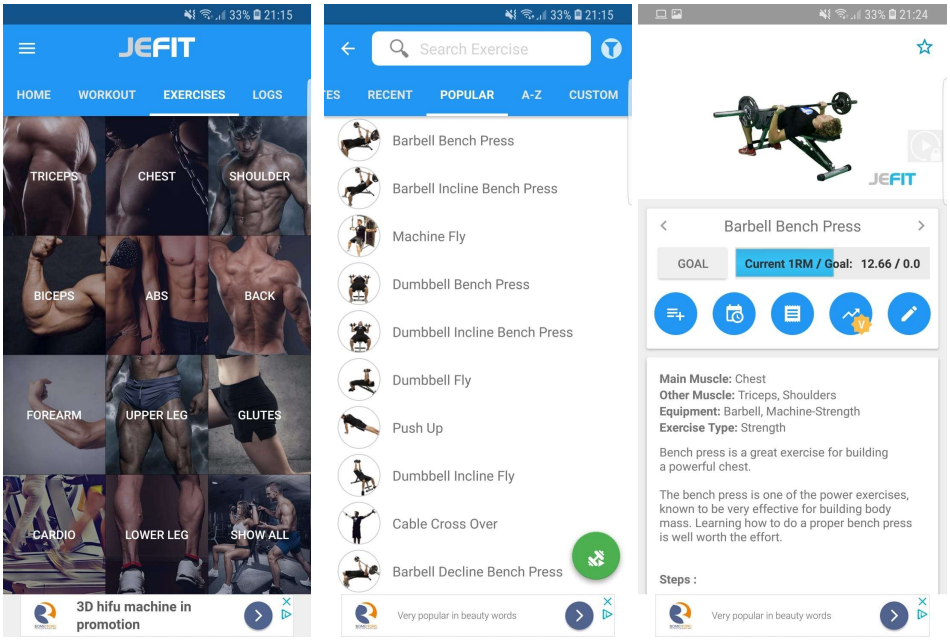

Figure 23 – Jefit's exercises

As depicted on Figure 23, the list of quick filters by muscle group, the exercise list, and information about a single exercise, can be seen, from the left to rightmost screenshot respectively. For each exercise, one can set a goal, add notes, set as favorite, watch the demonstrative video and also view the descriptive information. There's also the possibility to create exercises and filter them by other characteristics such as equipment, popular exercises, recent, etc.

Furthermore, there's an incorporated calendar, Figure 24, where one can see the logs of the past workouts, progress photos, notes, and body stat updates.

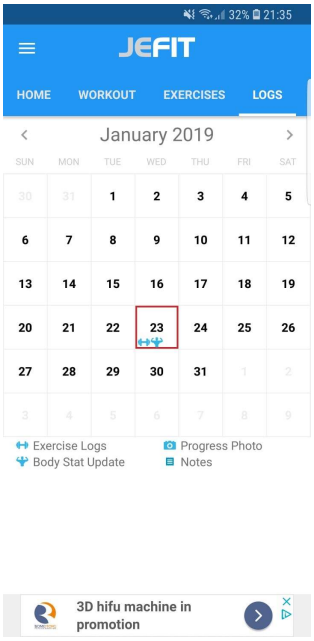

Figure 24 – Jefit's calendar

As a free feature, one can also check stats on training progress, send and receive messages, comment on other people's workouts (every workout is published to the feed) and customize certain characteristics of the app like disabling notifications, language, unit system, connect body measurements with Google Fit, among others.

4.7.1.2 Premium Main Features

Jefit is completely usable without a premium subscription, but if desired, with one, there are some complementary features offered. Amongst them is the accessibility to all premium workout plans and to the previously mentioned more than 1100 HD instructional videos of exercises.

Furthermore, there's also another feature only available with premium, which is the ability to swap an exercise automatically for another one. This is especially useful because sometimes a piece of equipment is being used or maybe the user doesn't feel like performing a particular exercise, and with this feature, they can simply swap it with another one, chosen by an algorithm.

Lastly, other premium features are such as: weekly progress reports, access to the web platform (with the same features of the app), unlimited cloud storage capacity, no adds, unlimited favorite exercises and routines and duplicating workouts and exercises.

4.7.1.3 Strengths

Undoubtedly, the biggest strength of Jefit is its calendar. Being able to pinpoint the exact dates on the calendar where things like progress photos or workout logs took place is very

advantageous and unique. Also, the filter options for the exercises are somewhat acceptable, given the fact that the app has more than 1300 exercises.

4.7.1.4 Weaknesses

Jefit also has its weaknesses, and one of them is the workout plan. The plan is simply a collection of up to seven isolated workouts that are meant to be repeated each week, there is no connection between weekly plans or even between its workouts. Even though it has more than some of the apps of the competition, it still falls short from a programming perspective.

Moreover, something that at first sight is not perceived as a weakness is the overwhelming number of exercises. Even with the maximum number of filters (muscle group and equipment needed), in most cases, there are still too many exercises, which for someone trying to find something can be problematic.

Also, at the first log in the app, an in-depth profile is outlined with the user's sex, age, preferred unit of measurement, training location, experience level, fitness goals, height and weight, and even though some of the information is required for default values, such as the units of measurement, age and sex, the others serve no apparent purpose. One might argue that for example the training location and the experience level would serve for the app to recommend workout plans but there's no recommendations whatsoever. For someone that defined that they workout on a gym there will still be home workouts to pick from, which can be great because it adds more versatility, but without a recommendation system, the profile outlined is just useless.

4.7.1.5 Price Plan[21]

In order to subscribe to premium [74], one has two options: to pay $6.99/month, or $3.33/month if a full year subscription is bought.

4.7.1.6 Final Notes

As a whole, Jefit is a decent app for not so serious lifters, since it provides a wide range of exercises and the ability to create simple weekly plans to follow. It also has useful features like the integrated calendar and the body measurements tracker, but unfortunately there's nothing that makes Jefit unique, in broad terms, it's just another exercise database with the ability to create and follow weekly exercise routines. The fact that a supposedly individual profile is outlined without any purpose at all makes the whole system a bit questionable. Furthermore, the number of exercises does seem a bit overwhelming – more is not always good.

Regardless, as already stated, for people that are just looking to workout at home or in the gym with no intent to carefully plan their routines or to measure progress, Jefit is an acceptable app.

[21] Values in dollars to avoid conversion. Data from February 20th, 2019

4.7.2 Fitbod

Fitbod [75] is everything an app should be – simple, attractive and useful. It's easy at first glance to mistake it for an underfeatured app, but that is not the case at all. Designed to automatically build daily workouts, based on individual preferences that can be easily customizable. Removing the strain of having to put together day-to-day workouts, Fitbod has the right algorithm for every type of goal, experience level, and even available equipment.

4.7.2.1 Free Main Features

As it was already stated, Fitbod is everything but underfeatured, and, most of its features are free. The main one is the ability to generate automatically a daily workout, based on a number of parameters, that can be, for the most part, customizable. On the first log of the app, the user enters information that will help the app's algorithm to suggest the most fit workouts. These are the aforementioned parameters, and can be listed as such: experience level, fitness goal, available equipment, last worked-out muscle groups and workout frequency.

These parameters work as a filter for the algorithm that builds the daily workouts. For example, as depicted on Figure 25, the user can select the list of equipment they have available, and the algorithm will only suggest exercises accordingly. Also, the user can select the group of muscles they want to focus on, and the algorithm will be applicable the same way it did with the equipment, only selecting exercises that work those specific muscles.

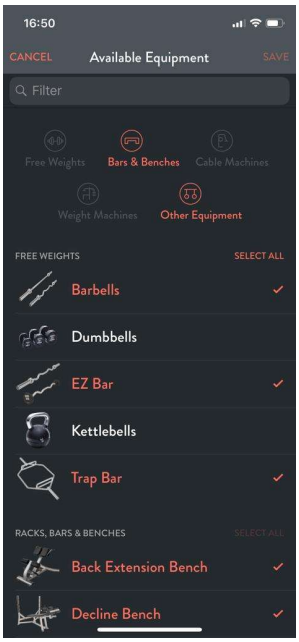

Figure 25 – Fitbod's Available Equipment Selection

Also, there's another variable to be considered by the algorithm when generating a workout, and that is the level of muscle tiredness. As depicted on Figure 26, the app has a dedicated screen to show the muscle recovery, that is used by the algorithm, and can be used even by the user, in order to decide which muscles are in need to rest and which ones can be trained. The

muscle recovery percentage is calculated automatically with the workouts performed, but can also be customized, as seen on the rightmost screenshot of the said figure.

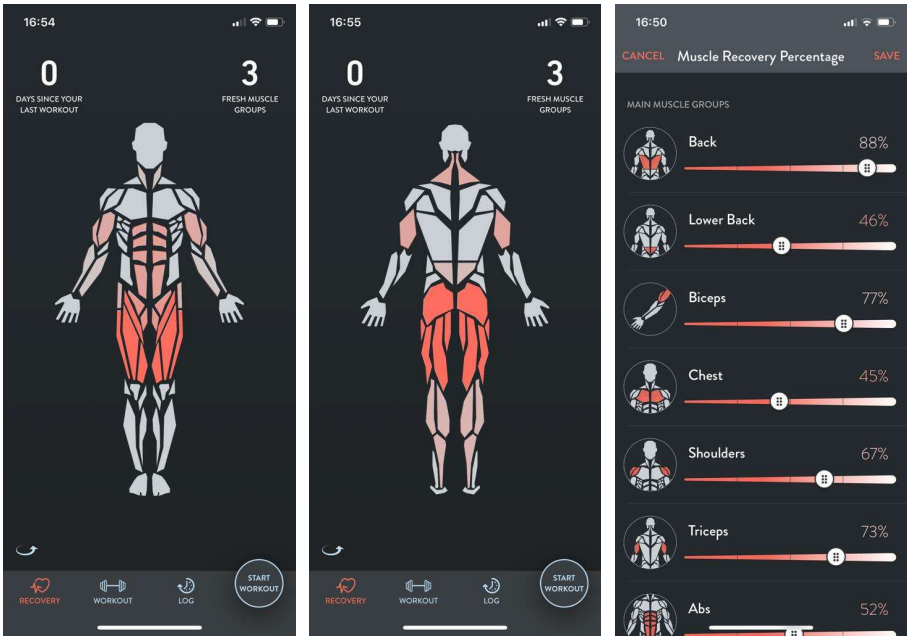

Figure 26 – Fitbod's Muscle Recovery

Moreover, still regarding the workouts, the user can create new workouts with one of a few options: generate a new workout with the app's algorithm, generate a workout that targets specific muscle groups, create a workout from scratch by selecting the desired exercises, or selecting basic workout splits, that emphasize different muscles/movements.

In addition, a user can also start a workout, in which they can enter the weight/repetitions performed and wait until the rest timer is over to start the next exercise/set. After the workout is finished, a summary is shown, with the amount of volume, calories burnt, and records achieved. The calculation for the calories burnt can be tuned in the settings, by connecting to Apple Health for personal body data.

Further free features can be listed as such: integration with Apple Watch, calendar logging past workouts, and informative videos and descriptions on exercises.

4.7.2.2 Premium Main Features

Even though most features are usable for free, there are still some that can only be acquired with a premium subscription. The main one is the ability to generate and perform a workout every day. This is because the free subscription only allows for three workouts for free. Also, a premium subscription will give access to more exercises and demonstrative videos and the ability to monitor their progress with stats.

4.7.2.3 Strengths

The main strength of Fitbod is most definitely its interface. Simple, intuitive and very well designed, it's built to catch the attention of the user and display its functionalities at their fullest. Furthermore, the ability to see body fatigue in the form of a heat map is very useful when deciding workouts.

4.7.2.4 Weaknesses

Even though Fitbod really looks like an app to take an example from, it still has some weaknesses. The most apparent one is the exercise selection from the algorithm. Even though they match up with the parameters selected by the user (muscle groups, equipment, etc.), it really does seem like the exercises are just picked at random from the list of exercises that match all the filters. Plus, the workout logs are somewhat incomplete, since one can only see the past workouts of the current week. If a user desired to consult the workout they did the previous month, they wouldn't been able to, which makes the calendar a bit pointless.

Furthermore, another thing that is undoubtedly a weakness is the inability to perform more than three workouts without a premium subscription. Even if the app wanted to be only usable by premium, it could give more than three workouts as a trial.

4.7.2.5 Price Plan[22]

The price plan for Fitbod has two different offers: either 61.99€ billed yearly, or 10.49€ billed every month.

4.7.2.6 Final Notes

There is no question whatsoever that Fitbod is a great app for people that want to work out on the go without having to worry about planning routines. Plus, the app's algorithm ability to filter out exercises that would hit still recovering muscle groups or exercises that would require equipment or experience that the user currently doesn't possess is very useful. That, allied to its great interface, makes it undoubtedly a great app to use for working out.

Nevertheless, the app's algorithm can't be called anything more than a filter, for reasons stated above. In such a simple system, it would be of great interest to have a true recommendation system, based on multiple factors such as previous exercise selections, previous feedback on routines, etc. Also, having to pay 10.49€ for a month is a bit over the top, for someone that can only perform three workouts as a trial.

All things considered, the success of the app shouldn't be overlooked, having a 4.8 rating on the Apple Store with almost 40.000 reviews [23], that is a clear sign that people are interested not only in useful but, more than anything, in an enjoyable experience of use.

[22] Data from February 20th, 2019
[23] Data from February 1st, 2019

4.8 Conclusion

As expected, the final section will be dedicated to summing the gathered information in a concluding manner. The evaluation of the app types and their apps will be presented, with the defined criteria for each in mind. Also, the envisioned system will be put to the test, facing the analyzed apps to assess the potential value it could introduce.

Table 3 – Summary of the strengths and weaknesses of the app types

App Type	**Strengths**	**Weaknesses**	**Target**
Gym Only	Direct connection with the personal trainer, nutritionist, and ability to schedule classes and physical evaluations.	Poor interface, few features overall.	Gym members.
PowerBuilding	Numerous tools for strength training (calculators, charts, stats), and multiple ways to program.	Poorly implemented timers.	Strength sport athletes, people seeking for strength training.
Athletic	Simplicity of use for athletic people.	Inexistence of a real progress-oriented tracking mechanism, and no customization.	Sports Athletes, people seeking for an athletic lifestyle.
General Lifting	Very versatile and diverse for most types of users.	Poor or inexistent profiling of users.	Average gym goers.

Table 3 consists in the summary of the information gathered throughout the analysis of the different types of apps. Each type, targeted at a specific type of active people, have their strengths and weaknesses and the ability to identify them will allow to build a system, ideally, suited for different types of users accordingly. It's relevant to mention that the strengths and weaknesses are judged in terms of the context of the app type, not in comparison to others. For example, even though besides the *PowerBuilding* app type, none of the others offer the ability to program a full-sized program, that can't be a weakness for them because it's a feature that mostly only people interested in strength training are looking for, and that is not their target.

In specific, one could say that the Gym Only apps' strengths lie on the ability to have a direct connection between the user and the gym they attend, by being able to schedule group classes, communicate with the personal trainer, nutritionist, among others. In contrast, since most gym

apps are not designed specifically for a gym but are sold to many, the general usability is feeble and the features insufficient.

Moreover, the weakness of *PowerBuilding* apps is on the poor implementation of timers. Timers are hugely important for people trying to maximize and perfect their strength/muscle-gain, and the fact that the apps designed for such goals do not focus on a good implementation of them is worrying. But, on the other hand, *PowerBuilding* apps have numerous useful tools for their target, ranging from plate calculators, to in-depth progress-oriented statistics.

Also, when it comes to Athletic apps, one of the things that really arises is the simplicity of use. The ability to start a workout and be done in 10 minutes with minimal interaction with a phone is definitely something that adds tremendous value to their target. However, that simplicity makes it so that there's virtually no useful progress-tracking mechanism to measure progress, and no ability to customize exercises or workouts, which is unfortunate.

Lastly, when it comes to General Lifting apps, their main issue is the fact that they try to suit everyone but either don't try to adapt the experience for different people through profiling, or they do it poorly. For example, it would be pretty unproductive to give a plate calculator for someone that is only looking to do bodyweight exercises. Plus, trying to suit everyone comes with another fault – overwhelming number of information (exercises, programs, etc.). Regardless, versatility, if well-executed, can be a great strength.

Table 4 – Apps' features summary

	AmazinGym	**Tripla Forma**	**Intensity**	**Strong**	**Nike Training Club**	**Freeletics**	**Jefit**	**Fitbod**
Profiling	No.	No.	No.	No.	Yes.	Yes.~	Yes.	Yes.
Create Workout	Yes.	Yes.	Yes.*	Yes.~	No.	No.	Yes.~	Yes.*
Start/Register Workout	Yes.	Yes.	Yes.	Yes.	Yes.	Yes.~	Yes.	Yes.*
Real-time Workout	Yes.	No.	No.	Yes.	Yes.	Yes.	Yes.	Yes.*
Workout Log	Yes.	Yes.	Yes.	Yes.	Yes.	Yes.	Yes.~	Yes.*
Workout Statistics	Yes.	No.	Yes.	Yes.*	No.	No.	Yes.*	Yes.~
Exercise Info	Yes.	Yes.	No.	Yes.	Yes.	Yes.~	Yes.~	Yes.~
Create Exercise	No.	No.	Yes.	Yes.	No.	No.	Yes.	No.
Create Training Plan	Yes.	Yes.	No.	No.	No.	No.	Yes.~	No.
Create Training Program	No.	No.	Yes.*	No.	No.	No.	No.	No.
Body Measurements Log	Yes.	Yes.	No.	Yes.~	No.	No.	Yes.~	No.
Body Measurements Statistics	No.	Yes.	No.	Yes.~	No.	No.	No.	No.
Progress Photos/Videos	No.	No.	Yes.*	No.	No.	No.	Yes.	No.

PT/Coach Communication	Yes.	Yes.	No.	No.	No.	No.	No.	No.
Multilanguage	No.	No.	No.	No.	No.	No.	Yes.	No.
Social	Yes.	Yes.	Yes.*	No.	Yes.	Yes.	Yes.	Yes.
Calculators	No.	No.	Yes.	Yes.*	No.	No.	No.	No.
Integrations	Yes.	No.	No.	Yes.	Yes.	Yes.	Yes.	Yes.
Export/Import Data	No.	No.	Yes.	Yes.	No.	No.	No.	Yes.
Web/Desktop Application	Yes.	No.	Yes.	No.	No.	No.	Yes.*	No.
AI-based Recommendations	No.	No.	No.	No.	No.	Yes.~	No.	No.
Number of Features	12/21	9/21	11/21	12/21	7/21	8/21	15/21	11/21

Table 4 showcases the information gathered throughout the individual analysis of each app, evaluating them based on the previously defined criteria. The number of features each app possesses from the ones presented is also discriminated, in order to objectively evaluate them. Even though this method could misjudge the app's intent, since obviously not all apps would want all the features presented, it's the most effective one to strongly position the envisioned system because the goal is to contain all of them (the features) in the most adaptive way, offering value to any type of user. Furthermore, the features marked with a "*" refer to features only accessible through a premium subscription and the ones with a "~" refer to poorly/questionably implemented features.

Moreover, one could say all apps fell a bit short on the evaluation, given that the one with the most features only had 15 and the one with fewer had 7, both out of 21 in total. This could obviously mean that the apps were, in general, designed to serve a specific purpose and not to "have it all", but, even though this may be true in some cases, in most, it is just reflection of an under-featured app. For example, Nike Training Club is an app designed for quick athletic workouts, so, being able to program a full-sized program probably wouldn't offer much value to its users, but the fact that it doesn't offer things like multilanguage, customization, progress-oriented features, and so on, it is just considered to be under-featured.

Specifically, there are a few features that deserve to be pointed out, and the first one is the profiling one. The only apps that had some sort of profiling were both the Athletic type ones, and General Lifting, which makes sense because for quick workouts, there are tons of possibilities and the ability to profile users and adapt itself accordingly is very important, and also, for General Lifting apps, designed to suit the necessities of various types of users, profiling is essential. The effectiveness of the said profiling was already described thoroughly, but on a final note, it felt a bit short. This profiling is the backbone of "intelligent" or "smart" recommendations, but only one app had AI-based recommendations, and even that was questionable, as was also covered before.

Additionally, in terms of workouts and workout planning, only the Athletic apps didn't allow for the creation of a workout, which is unfortunate. Plus, only Gym Only apps and Jefit allowed for the creation of a training plan, and just one of the *PowerBuilding* apps, Intensity, offered the ability to program a full-sized program, which justifies why the app was chosen in spite of not meeting any of the selection criteria. The fact that an app, such as Strong, has a rating 0.5/5 bigger than Intensity, with almost 120 times more the number of ratings but doesn't offer such a specific and needed feature is questionable at best [24].

In conclusion, the whole process of this chapter was crucial to the development of the further ones, since it allowed to grasp what constituted value in the context of the investigation. The establishment of selection and evaluation criteria made the analysis of the apps and their types much more objective and rigorous. Plus, understanding the strengths and weaknesses of each will be useful when deciding what to replicate and what to improve.

[24] Data from February 1st, 2019

5 Design

This chapter will be dedicated to the exposition of two big sections – requirements engineering and the system's architecture.

The first section, Requirements Engineering, consists in the design of software requirements, in the form of functional and non-functional requirements and of the different stakeholders and interveners that are related to them. The second one, Architecture, is composed by the presentation and description of the different proposed architectures to answer the designed requirements, and also by a comparison between them in order to pick the best one in the present context.

5.1 Requirements Engineering

This section will be dedicated to the "process of conforming engineering designs to a set of core software requirements", also known as *Requirements Engineering*. In the next sub-sections, the study and documentation of the said process will be presented. It's also important to mention that the requirements were, in part, based off of information gathered through the execution of a "*Usage Analysis of Gym/Fitness Apps*", in which nearly 90 different users offered their opinion in regards to their personal experience with fitness mobile applications and what they deemed valuable. This inquiry and further information can be found in the APPENDIX A.

5.1.1 Stakeholders

A stakeholder, according to the *Business Dictionary* [76] can be defined as an entity (person, group or organization) that has interest in an organization. It can directly or indirectly affect or be affected by the organization's objectives, policies and overall actions. Because of this, it becomes important to define the stakeholders of this project, given that they will be affected with the outcome of it.

The stakeholders of this project and their interests are as follows:

- ***Gyms***: Interest in offering a suitable platform to their members.
- ***Gymgoer (fitness practitioner):*** Interest in using the platform for all the offered features.
- ***Personal Trainer/Coach:*** Interest in having a suitable platform to monitor their trainees' progress.
- ***Nutritionist:*** Interest in using the platform for monitor their clients' progress.
- ***System Administrator:*** Interest in managing the system.

5.1.2 Interveners

The intervener is a role played by an entity, usually a person, that interacts with the system and exploits its functionalities. Commonly, the interveners are the stakeholders, since most of them will directly use the system, but it's not rare to have some stakeholders that are not interveners. In any case, they're presented next:

- ***Non-registered user:*** This is a person who's not yet registered in the system and needs to do so in order to access any other features.
- ***User (gymgoer):*** This is the person that can use the system and access most of its features, depending on some factors.
- ***Personal trainer/Coach:*** This is the person responsible for the features related with trainee monitoring.
- ***Nutritionist:*** This is the person responsible for all the nutrition-related features (i.e. monitoring clients).
- ***System Administrator:*** This is the person responsible for the system as a whole, introducing new data (i.e. scientific articles) and maintaining it to assure it's running smoothly.

5.1.3 Functional Requirements (Use Cases)

This section will contain information regarding the functional requirements of the system, in the form of use cases. According to *Agile Modeling* [77], a use case diagram "overview[s] the usage requirements for a system. They are useful for presentations to management and/or project stakeholders […]" [77]. They depict the use cases of the system, which are "a sequence of actions that provide something of measurable value to an actor […]" [77], and the associations between them and the said actors, which can be viewed as something or someone that play a role in the system.

Figure 27 – Use Case Diagram

In Figure 27, the use case diagram of the system is depicted, and in the following Table 5 the priority distribution associated with each one is presented, so that during implementation some can be prioritized over others, if due to time restrictions not everyone can be implemented.

Table 5 – Use Cases Priority

Priority	**Use Cases**
High	US02; US03; US04; US06; US07; US08; US10; US13; US15.
Medium	US01; US11; US16; US17; PT01;
Low	US05; US09; US12; US14; NT01; AD01; AD02.

In the following sub-sections will unveil specific details about each use case in it. It is also important to mention that some use cases are too broad and can possibly be divided into sub-use cases in further iterations.

5.1.3.1 US01: View current training Plan/Program

The user requests the system to view the current training plan/program they are following. The system presents the requested information.

5.1.3.2 US02: View training Plan/Program templates

The user requests the system to present the list of template training plans/programs. The system presents the requested list. The user can then select an individual training plan/program and the system presents the specific information about it.

5.1.3.3 US03: Manage training Plan/Program

The user requests the system to create a new training plan/workout or edit an existing one. The system provides the user with a suitable interface to create/edit a training plan/program and then, if changes were made, the system requests confirmation and saves the information.

5.1.3.4 US04: Manage Body Measurements

The user requests the system to configure their personal body measurements, to create a new entry or to edit an existing one. The system provides the user with a suitable interface to perform the desired task and then requests confirmation and saves the information.

5.1.3.5 US05: Add favorite training Plan/Program

The user requests the system to set a training Plan/Program as favorite. The system sets the selected information as favorite and saves the information.

5.1.3.6 US06: Recommend training Plan

The user requests the system to recommend a training Plan. Based on the user's profile, acquired data, and other variables, the system offers a suitable training plan recommendation.

5.1.3.7 US07: Recommend Workout

The user requests the system to recommend a Workout. Based on the user's profile, acquired data, and other variables, the system offers a suitable workout recommendation.

5.1.3.8 US08: View training Logs

The user requests the system to present a calendar with information regarding their training logs. The system presents the requested information.

5.1.3.9 US09: Share training Plan/Program

The user requests the system to share a training Plan/Program via social networks. The system shares the selected information in the preferred social network/s.

5.1.3.10 US10: View Statistics

The user requests the system to present statistics regarding multiple variables like training progress, body measurements, and others. The system presents the user with the requested statistics.

5.1.3.11 US11: Monitor progress

The user requests the system to present information regarding progress statistical analysis. The system presents the requested analysis in order to be monitored by the user.

5.1.3.12 US12: Interact with PT/Coach

The user requests the system to interact with their personal trainer/coach via direct messaging. The system facilitates the interaction offering the user with a suitable interface for communicating.

5.1.3.13 US13: Interact with PVA (Personal Virtual Assistant)

The user requests the system to interact with the Personal Virtual Assistant via *chat bot*. The system facilitates the interaction offering the user with a suitable interface for communicating.

5.1.3.14 US14: Check social feed

The user requests the system to present the social feed. The system presents the user with the requested information.

5.1.3.15 US15: Change Language

The user requests the system to change the system's language. The system requests information regarding the new language to be changed to. The user enters the desired language. The system updates and saves the information.

5.1.3.16 US16: Change Unit System

The user requests the system to change the preferred unit system. The system requests information regarding the new unit system. The user enters the new desired system. The system updates and saves the information.

5.1.3.17 US17: Manage Available Equipment

The user requests the system to change the available equipment list. The system presents the user with the desired list. The user makes the desired changes and requests the new information to be saved. The system confirms the changes and saves them.

5.1.3.18 PT01: Monitor trainees

The personal trainer requests the system to present the list of current trainees. The system presents the requested list. The personal trainer selects an individual trainee and requests specific information regarding them. The system presents the personal trainer with the requested information.

5.1.3.19 NT01: Monitor clients

The nutritionist requests the system to present the list of their clients. The system presents the requested list. The nutritionist selects and individual trainee and requests specific information regarding them. The system presents the nutritionist with the requested information.

5.1.3.20 AD01: Manage predefined training Plan/Program

The administrator requests the system to manage the predefined training Plans/Programs. The system presents the administrator with the current predefined training Plans/Programs. The administrator can then create a new training Plan/Program or edit existing ones. If changes were made, the system requests confirmation and saves the information.

5.1.3.21 AD02: Manage scientific articles

The administrator requests the system to manage scientific articles. The system presents the administrator with the current scientific articles. The administrator can then add a new scientific article or edit existing ones. If changes were made, the system requests confirmation and saves the information.

5.1.4 Non-Functional Requirements (FURPS+)

After presenting the functional requirements, that simply describe what the system should do, there's the need to also showcase the non-functional requirements, that describe how the system works [78], and, one might say, they describe the system's attributes. To present and classify these attributes, the *FURPS+* was used. *FURPS+* [79] is a system to classify requirements and is represented by the categories presented next.

5.1.4.1 Functionalities

- ***Authentication:*** The use of the system requires a pre-authentication by the user.
- ***Security:*** To any system, security is a crucial aspect, and as such, there are two major characteristics that need to be in place:
    - ***Authenticity:*** Ensure the entities are who they claim to be, to separate different access roles.
    - ***Confidentiality:*** Ensure data is confidential, through encryption methods.

5.1.4.2 Usability

- ***User Interaction:*** The interaction with the user must be simple, intuitive, and completely adapted to the respective environment (i.e. used device).
- ***Help:*** The system must provide suitable and contextualized help to the task the user is performing.
- ***Interface:*** It's desirable for the interface to be appealing and clear.
- ***Error Prevention:*** The system must be "forgiving" in the sense that should prevent user mistakes and treat them accordingly.

5.1.4.3 Reliability

- ***Predictability:*** The system should be reliable, that is, it should be free of technical errors.
- ***Fault Tolerance:*** The system should be error-tolerant to protect the user from unintentional errors.

5.1.4.4 Performance

- ***Response Time:*** The system's response time should be fast, to provide quick access to data.
- ***Availability:*** The system should have a very high availability rate.
- ***Memory Usage:*** The CPU and memory usage should be fairly low during usage.
- ***Capacity Load:*** The system's capacity load should be very high because there's a great quantity of data being handled.

5.1.4.5 Supportability

- ***Portability:*** The system should be available for Android and iOS operating systems.
- ***Testability:*** The system should be easily testable in order to provide high confidence about correctness.
- ***Maintainability:*** The system should have high maintainability, in order to allow future requirements and/or repairs.
- ***Localizability:*** The system should support multiple languages.

5.2 Architecture

This section is dedicated to the presentation of the different proposed architectures, as well as showcasing the chosen one, properly justified and detailed.

5.2.1 Domain Model

A domain model is a conceptual model, organized and structured around the knowledge of a problem. It "[…] should represent the vocabulary and key concepts of the problem domain and it should identify the relationships among all entities [within its scope]." [82] One of the breakdowns that projects usually suffer are due to misconceptions or misunderstandings of concepts, so, the domain model and its key concepts and definitions should be understandable by everyone involved (programmers, team leaders, clients, etc.).

With that, for a better understanding of the proposed solution and to respond to the specified requirements, a domain model was designed, depicted in the next Figure 28.

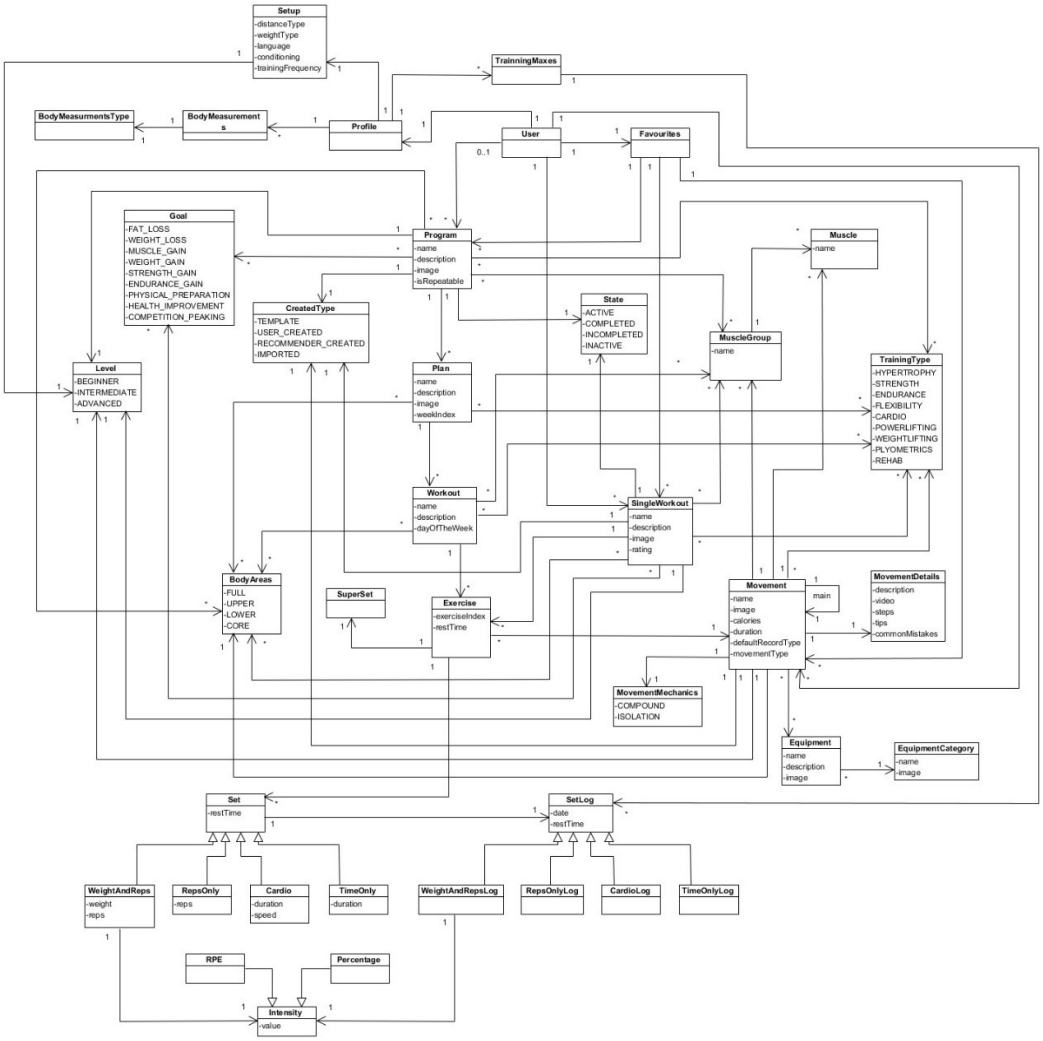

Figure 28 – Domain model

In the designed model, there are a great deal of involved entities, so, to sum up, the most important and crucial ones will be briefly detailed:

- ***User***: Entity that benefits from the use of the functionalities granted by the system. They can have their own *Programs* and *Single Workouts* or manipulate already existing ones offered by the system. They also own a *Profile* in which their setup information lies (i.e. language, units of measure, etc.), among other things like their *Body Measurements* and *Training Maxes*.

- ***Program***: A single *Program* can have multiple *Plans*. This entity is characterized with different other entities, such as a list of associated *Goals*, a *State*, a list of *Muscle Groups*, among others. A *Program* can be seen as an aggregate of different training weeks.

- ***Plan***: A *Plan* can have multiple *Workouts* and it is characterized by the *Body Areas* it focuses on, *Training Types* and others. It can exist as part of a *Program* or as a standalone *Plan*. A *Plan* can be seen as a training week, with different training days.

- ***Workout***: A *Workout* might be regarded as a training day, also characterized with different entities.

- ***Single Workout***: The difference between a *Workout* and a *Single Workout* is that the latter exists in itself without being part of any *Plan*. This is beneficial for many reasons, such as allowing a *User* to perform a *Workout* without the need to "subscribe" to a full-sized training *Plan/Program*.

- ***Exercise***: This entity depicts a fitness is characterized by having a *Movement* and a list of *Sets*. It can also be a *Super Set* which simply means it can be seen not as a single exercise but an aggregate of multiple ones, executed in sequence.

- ***Movement***: A *Movement* is one of the most complex entities in terms of association with other ones. It is defined by multiple characteristics like its details (*Movement Details*), its *Movement Mechanic* and the list of necessary *Equipment* to conduct the said *Movement*. It is also characterized by multiple other entities such as a list of *Muscle Groups*, *Training Types*, *Body Areas*, and others.

- ***Set***: This entity represents a *Set* of an *Exercise*, which is the number of cycles of repetitions for a given *Exercise* and can be represented by any of the four types:

    - ***Weight and Reps*:** Defined by a given weight and the number of associated repetitions. It can also have an *Intensity* attribute, which can be in the form of *RPE* or a *Percentage*.

    - ***Reps Only*:** Defined only by the number of repetitions needed.

    - ***Cardio***: Defined by the duration and average speed (for cardiovascular exercises such as running or biking).

    - ***Time Only***: Defined only by the duration.

- ***Set Log***: This entity is very crucial in the log of information. Every *Set* has an associated *Set Log* and they both possess the same already-described attributes. One might see the *Set* as a prediction of what it is supposed to be done, and the *Set Log* as a record of what was effectively performed (for example, in terms of repetitions).

5.2.2 Proposed Architectures

In order to find a suitable architecture, there were designed three different ones, each with its own characteristics. They will be presented next, through the use of a *Logical View*, as well as a brief explanation of the said characteristics they possess. The *Logical View* consists in a "conceptual organization of the software elements in terms of the most important layers, subsystems, packages, [etc.]." [80] It's logical because there's no relationship with the deployment by operative systems, processes or physical nodes (computers). [81]

It's also important to mention that all the propositions have some common characteristics, and only the differences between them will be discussed in the next sections. All the solutions have four major common components:

- ***FitnessBackOffice:*** This is the back office of the system, which comprises the software used to administer operations that are not related to any direct customer interface. [82]

- ***FitnessWebAPP:*** In contrast to the back office, this is one of the front offices, which is an application that directly interacts with the customer. [83] This specific one refers to the web application.

- ***FitnessMobileAPP:*** This is also a front office, as described in the prior component, the difference is that this one refers to the mobile application.

- ***FitnessRecommenderSystem:*** This component is responsible for the manipulation of user data in order to produce recommendations.

- ***ChatBot:*** This is the component responsible for handling the Personal Virtual Assistant's business logic.

- ***LUIS API:*** Machine-learning-based service responsible to handle communication with the *ChatBot*.

5.2.2.1 Alternative 1

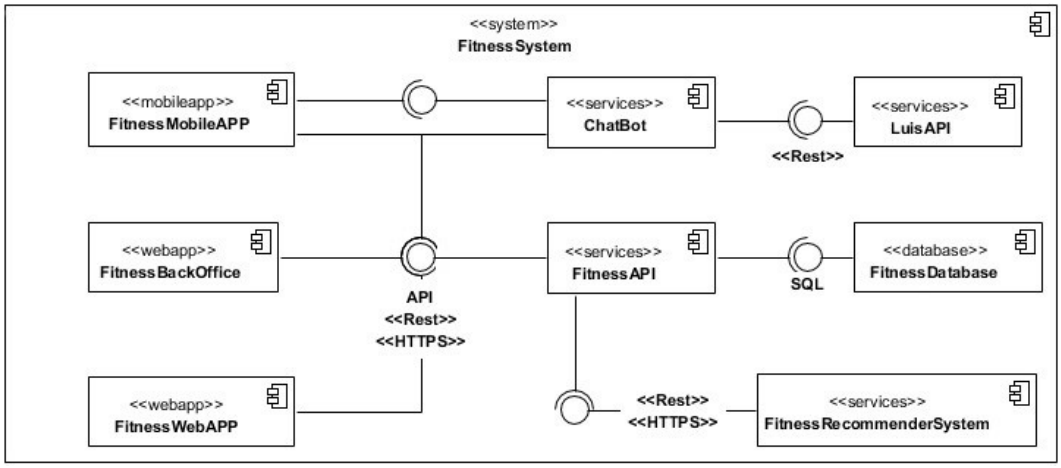

Figure 29 – Logical View of the first proposed architecture

The first proposed architecture, depicted in Figure 29, is characterized by the existence of an *API* (*FitnessAPI*) that handles all the operations between the other components and the database. It provides an interface that all the other components use in order to communicate with the database and with each other. Plus, there's a single database that is accessed by the *API* and the *API* only. This allows for multiple advantages like efficiency due to the easiness of publishing, integration, personalization, adaptation, and others. [84] Also, the fact that only one component is responsible for the orchestration of operations between the database and the others is a major advantage.

5.2.2.2 Alternative 2

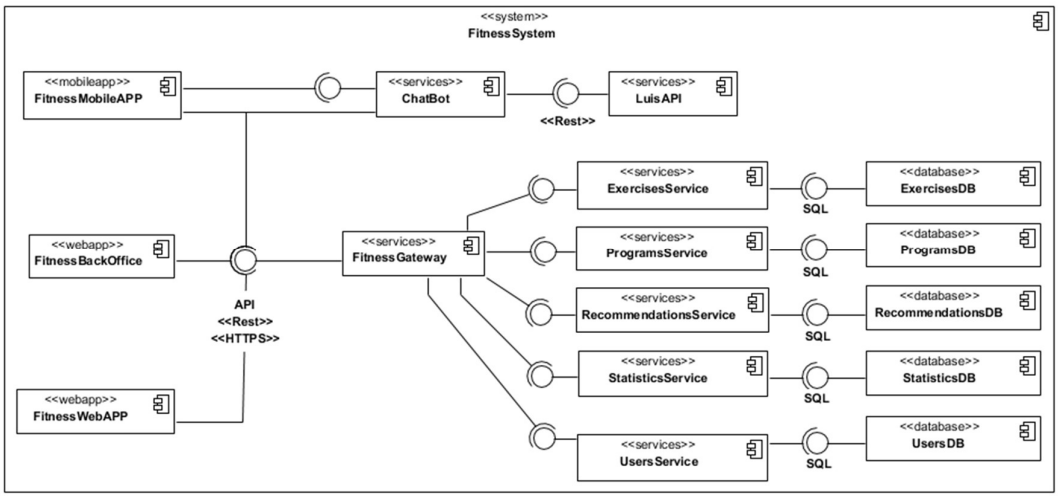

Figure 30 – Logical View of the second proposed architecture

The second proposed architecture, depicted in Figure 30, is characterized by the implementation of a *microservices architectural pattern* [85]. This pattern "structures the application as a set of loosely coupled, collaborating services […]" [85] which has multiple benefits such as better testability, because the services are smaller and faster to test, better deployability, because services can be deployed independently, improved fault isolation, and because each microservice is relatively small, it's easier for developers to understand, the IDE is faster and the application starts faster, which improves productivity. Also, it "[e]liminates any long-term commitment to a technology […]. When developing a new service, you can pick a new technology […]." [85]

As so, there's a new component named *FitnessGateway* that handles the communication between the other components and all the microservices. The created microservices can be seen in the image, and all encapsulate different responsibilities. Also, each one has their own database, in order for them to be completely isolated from each other. The created microservices and their responsibilities can be presented as:

- ***ExercisesService:*** responsible for all the exercise-related business logic (i.e. add a new exercise and check information of a specific exercise).

- ***ProgramsService:*** responsible for all the program-related business logic (i.e. create a new training plan/program).

- ***RecommendationsService:*** responsible for all the recommendation-related business logic (i.e. recommend a workout).

- ***StatisticsService:*** responsible for all the statistic-related business logic (i.e. generate body measurements statistics).

- ***UsersService:*** responsible for all the user-related business logic (i.e. logging of a user).

It is important to mention that not all the needed microservices to make the system work are represented, only the most important ones are depicted, for organization purposes. In case this is the selected architecture, a new, more in-depth solution needs to be formulated.

5.2.2.3 Alternative 3

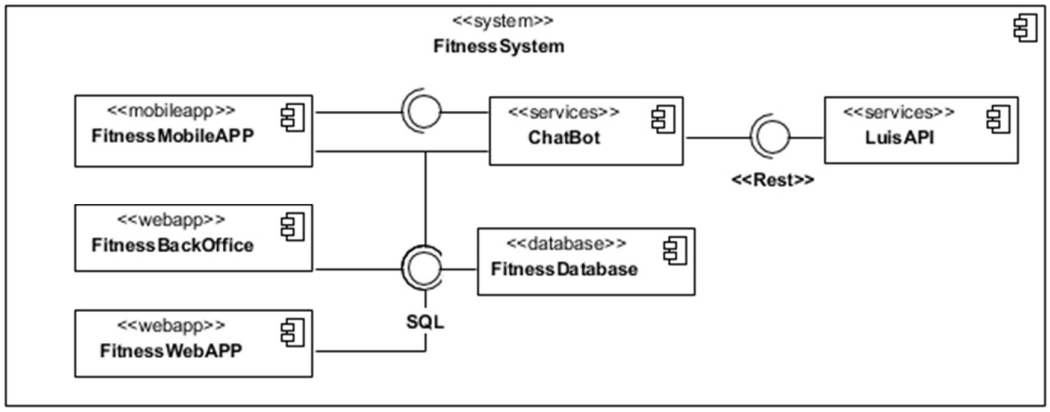

Figure 31 – Logical View of the third proposed architecture

The third proposed architecture, depicted in Figure 31, is the simplest of them all in terms of number of used components. The difference from this architecture and the other ones is the inexistence of an *API* or some sort of operation redirector (i.e. *Gateway*). In this solution, the business logic is encapsulated throughout the existing components, and they all communicate directly with the database. This is undoubtedly an easier solution to implement, dude to the reduced number of components and connections.

5.2.3 Implemented Architecture

Having described all the different proposed solutions, it's now time to evaluate each one's strengths and weaknesses and pick the one with the greatest added value. In order to do so, a table was built, as follows.

Table 6 – Proposed Architectures Evaluation

Alternative	**Strengths**	**Weaknesses**
Alternative 1	Automation, flexibility of delivery, efficiency, personalization, adaptation, and others. [84] Reduced complexity of development.	Low maintainability on high scale. Hard to distribute work across development team.
Alternative 2	Better testability, deployability, and overall development. Improved fault isolation. [85]	Development and deployment increased complexity. Increased memory consumption. [85]
Alternative 3	Easy to implement due to reduced number of components.	Deficient distribution of responsibilities. Low maintainability.

Table 6 summarizes the information gathered in the previous section regarding the strengths and weaknesses of the different proposed architectures. This information will serve as justification to choose a preferred one to implement. From the information gathered, alternative number 3 can be ruled out – the ease of implementation doesn't overweight the deficiency of responsibility distribution and the very low maintainability. Also, the fact that there's no gateway to handle communication with the database is a severe disadvantage for almost any system, due to, among other things, data concurrency.

Thus, the final decision comes from picking the best suited architecture out of alternative 1 and 2. Simply put, alternative number 2 brings the best advantages, in the form of improved testability, development, and overall scalability. The issue with this alternative is that it's extremely complex for the project at hands. This alternative is indicated to large development teams, where developers can work on different services without conflicting with each other, and where the system is supposed to scale significantly. Because of that, this is not the best alternative, since the development team is small (2 developers) and the added complexity does not overweight the benefits it brings. Plus, the increased memory consumption and the deployment increased difficulty is enough reasons not to pick this alternative in the current context.

As such, alternative number 1, by process of elimination, is the best one. Simple enough to not cause any development and deployment trouble and robust enough to meet the needed demands for the desired system. The only disadvantages of this alternative are not really an

issue for this context because the team is not large enough to be hard to distribute work, and the scale won't be big enough to have maintainability problems.

5.2.4 Detailed Implemented Architecture

Having decided an architecture to implement, it is now time to detail with more thoroughly, more specifically in terms of its individual components and also determine how the overall system will be deployed, through the use of a *Deployment View*.

Accordingly, in the next sub-sections, the different architecture's components and the deployment of the system will be detailed.

5.2.4.1 FitnessAPI

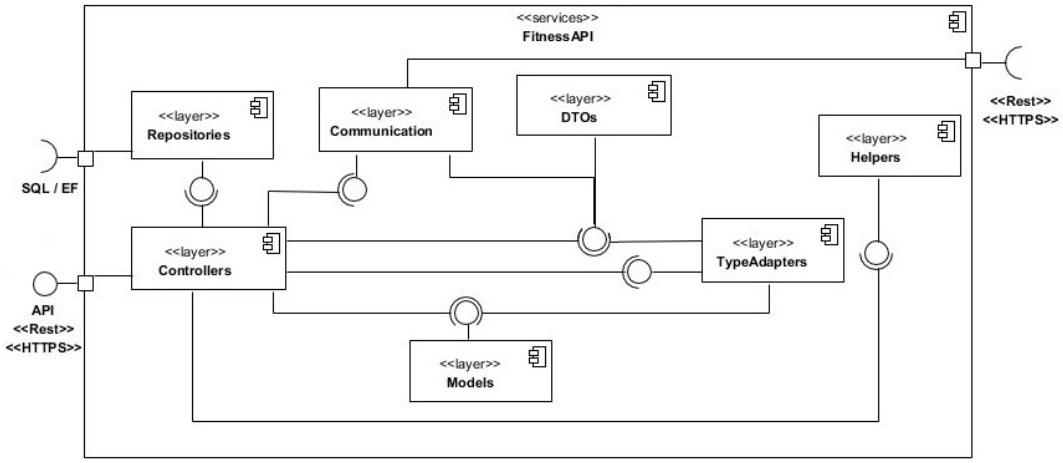

Figure 32 – API's Logical View

The previous Figure 32 depicts the logical view of the API with all its inner layers. The interface to which the other components access, is provided by the *Controllers*, where all the calls from outside go through before reaching the other layers. This layer is responsible for controlling the flow of the execution. It generates and manipulates data through actions that are then returned as results to the respective requests. The other layers' responsibilities can be described as follows:

- ***Models***: This layer is where all the different models of the system lie, which are responsible for handling business logic.
- ***DTOs***: Data transfer object, it carries encapsulated data between subsystems.
- ***Communication***: Responsible to handle all communication with external services.
- ***Repositories***: Responsible for abstracting the persistence of objects. It contains all logic related to persistence, as well as mapping between tables and objects.

- ***Helpers***: Layer responsible for bestowing classes that allow for a good working flow of the system, for example, conversions between units of measure, dictionary functions, etc.

- ***TypeAdapters***: Layer responsible for the conversion between *DTOs* and *Models* and vice versa.

5.2.4.2 FitnessMobileAPP

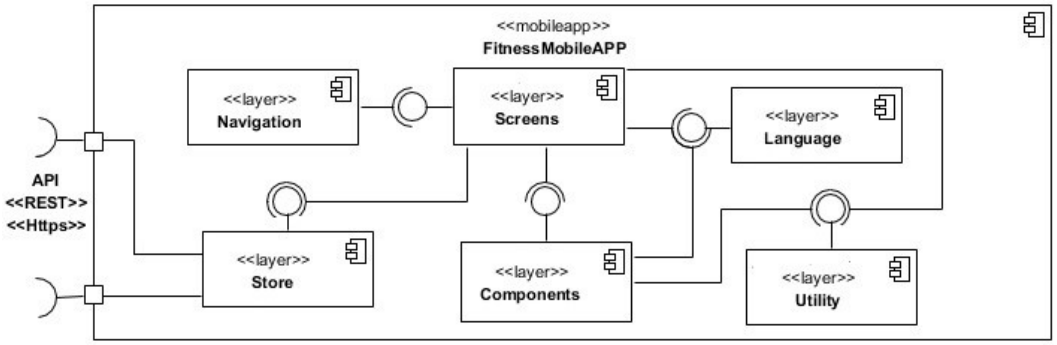

Figure 33 – Mobile App's Logical View

Figure 33 portrays the inner layers of the Mobile App component, and they can be presented as follows:

- ***Store***: Responsible for communicating with the server and managing the application's state.

- ***Screens***: This layer contains all the screens that are presented to the user. The logic behind the construction of the user interface lies in this component.

- ***Navigation***: Responsible for declaring the screens and controlling de flow between them.

- ***Components***: It is in this layer where all the reusable UI components reside.

- ***Utility***: Responsible for containing different functions and constants that support the good flow of the application.

- ***Language***: This layer contains the language component of the mobile app. Responsible for controlling the information's language that's being displayed to the user, depending on their preferences.

5.2.4.3 *FitnessRecommenderSystem*

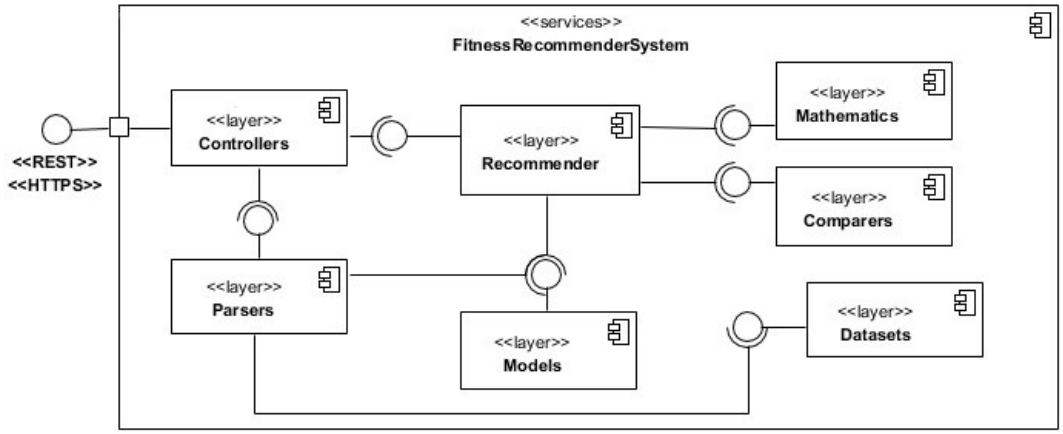

Figure 34 – Recommender System's Logical View

The Recommender System's component is being detailed in Figure 34. This component only communicates with the API, and all the logic is processed within itself. The *Controller* receives the requests from the API and then delegates the rest of the process to other layers with different responsibilities, presented next:

- ***Recommender***: This layer is where the different recommender algorithms are. These are the ones responsible to handle all the received information and produce recommendations accordingly.

- ***Parsers***: The *Parsers* consist in blocks of code (within their respective classes) that handle the parsing of information (i.e. importing and mapping of datasets).

- ***Models***: This layer is where all the different models of the system lie.

- ***Mathematics***: In this component, different mathematic-related tasks are conducted, such as the calculation of the *dot product* or *SVD*.

- ***Comparers***: Regarding the computation of recommendations, there's always a need to compare users in order to obtain the similarity between them, and it is in the *Comparers* layer where all these different techniques lie.

- ***Datasets***: There's no recommendation without training *Datasets*, which is the responsibility of this layer – to hold different *Datasets*.

5.2.4.4 *FitnessBackOffice* & *FitnessWebAPP*

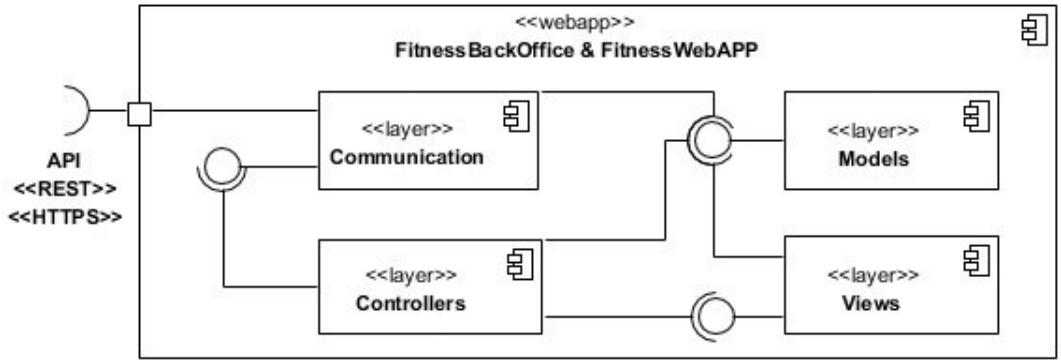

Figure 35 – *FitnessBackOffice* & *FitnessWebApp*'s Logical View

The system's web application and back office is depicted in Figure 35, and its components can be described as such:

- ***Communication***: Responsible to handle all communication with external services.
- ***Controllers***: Responsible for controlling the flow of the application and generating and manipulating data through actions that are then returned as results to specific requests.
- ***Models***: This layer is where all the different models of the system lie, which are responsible for handling business logic.
- ***Views***: Layer where all the views presented to the user reside.

5.2.4.5 System's Deployment

A deployment view [89] is an UML diagram used to view and describe the topology of the physical components of a system. In other others, "where the software components are deployed." [89] This view is very useful and important because it takes into account primarily the non-functional requirements of the system such as performance, scalability, availability and reliability. [90]

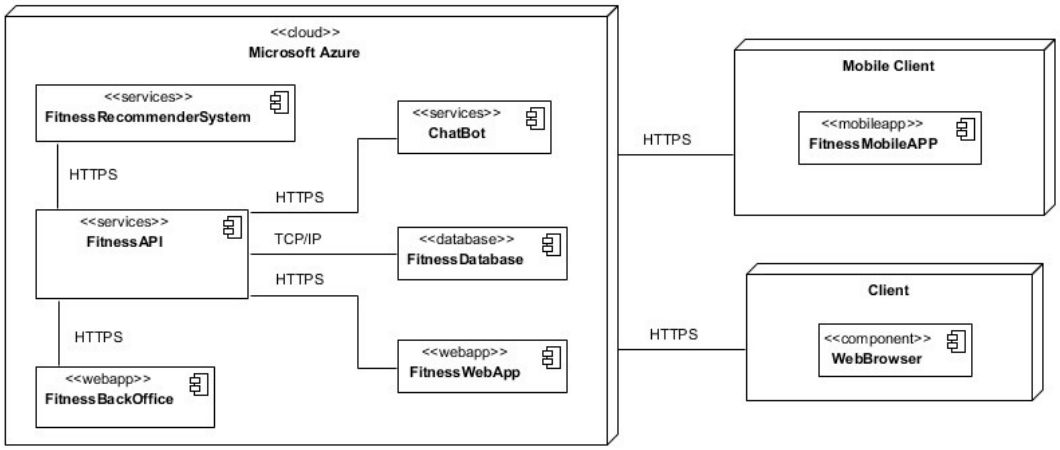

Figure 36 – System's Deployment View

The artifact produced, correspondent to the deployment view is depicted in Figure 36. All the back-end components are deployed into *Microsoft Azure*'s cloud, providing availability, security and also disaster recovery, given that everything is safely stored on *Azure*'s servers. Furthermore, from the components that are not on the cloud, two can be named - *FitnessMobileApp* and *WebBrowser*. These refer to the mobile app and the web version respectively. Additionally, the *WebBrowser* contains the already presented *FitnessWebApp* and *FitnessBackOffice*, for users and system administrators respectively.

As a side note, it is important to mention that the *LuisAPI* component, depicted in Figure 29 is not included in the deployment view due to the fact that it is a *Microsoft* service, having no deployment needs.

6 Implementation

In this chapter, all the implementation details will be presented, from the development of the Recommender System, to the Mobile Application, Personal Virtual Assistant, the Server Application and the Personal Virtual Assistant. Furthermore, the specific implemented uses cases will also be presented and detailed, as well as the Non-Functional Requirements. To conclude, the tests to which the system was subject are bestowed.

6.1 Recommender System

This section is dedicated to the detailing of the implementation of the Recommender System, which initially was predicted to be just an intelligent one, but through the course of its implementation, another one had to be devised, as it will be explored next.

6.1.1 Intelligent Recommender

A system is deemed intelligent if it has the capacity to learn from experience and the ability to adapt according to the available data. [99] Specifically, regarding recommender systems, if it can make recommendations based on some learning-process and accommodate itself to the data that is being fed, then it can be considered a smart or intelligent recommender.

The idea to implement such system came from the notion that many users of the currently being developed system will need some sort of help and assistance in their training. Having in place a system that supports them in their fitness life by making smart recommendations, based on their profile, on the environment (other users) and their overall history is highly desirable.

To implement such system, the *Matrix Factorization* technique was selected. From the presented techniques, it is the only one that is based on self-improvement and learning, since it adjusts its model to the data and to the calculated error, as it will be furtherly detailed in the next sub-sections.

Furthermore, in order to add some sort of personalization and differentiation it is also desirable to complement the intelligent recommendations provided by *Matrix Factorization* with of other techniques'. These will be *User-Based Collaborative Filtering* and *Item-Based Collaborative Filtering*, already described in the State-of-the-art chapter. To do so, another already covered technique will need to be employed, the *Hybrid-approach*, because it offers the mechanism to use multiple recommendation techniques at the same time.

The next sub-sections will be dedicated to exploring and detailing the implementation of the intelligent recommender system, with all its intricacies. It is also important to mention that the foundation for the present system's implementation is based in an article named "*Building a Recommendation Engine in C#*" [102], by *Scott Clayton*, which won the second prize in the *Best C# Article of March 2018* and the first prize in *The Machine Learning and Artificial Intelligence Challenge*.

6.1.1.1 Data Importation and Modeling

In this section, the process of importing and modeling data will be detailed. As a first note, it is important to mention that the data fed to the system is not context accurate. Meaning that, the information modeled and used to make recommendations is not workout-related and, therefore, not useful in practical terms. The reason for this is due to the fact that, for an intelligent recommender to work and be decently accurate at all, it needs a great amount of initial information, and the unfortunate reality of this project's situation is that there's not feasible way to gather sufficient and meaningful context accurate information with the time restrictions in place.

Consequently, a way to circumvent this hindrance had to be devised. The solution then became to use similar data structure-wise, to still maintain a system able to support context accurate information at any point in time that's available and still be able to test the implemented algorithm with data that is not fabricated. This can also be seen as an advantage because it is easier to test the results of the system by using real, verifiable data. This way, the system's recommendations can be put to test and the respective algorithm validated.

Keeping this in mind, it is of high importance to use a reliable dataset – if the information cannot be contextually true, at least it should be trustworthy in its own context to make valid conclusions on the system's integrity.

The selected dataset is provided by *GroupLens Research*[25], that gathered rating datasets from the *MovieLens*[26] website and made them available, especially for research and investigation purposes. Literature [103] supports that this dataset is used widely across education, research and industry. With hundreds of thousands of downloads each year, "reflecting [its] use in popular press programming books, traditional and online courses, and software." [103] It is heavily referenced in literature, with several hundreds of publications[27] and more than 16.000 results on *Google Scholar*[28].

[25] https://grouplens.org/datasets/movielens/
[26] https://movielens.org/
[27] https://grouplens.org/publications/
[28] https://scholar.google.com/

The structure of the dataset aligns almost perfectly with the desired one's, only requiring a few adjustments. For this reason, and because it is very extensive and trustworthy [103], it makes for an excellent choice.

Regarding the specific importation and modeling of the information, first, the relevant entities are imported – *users*, *items*, *ratings*, *tags*.

The users and ratings are imported simultaneously, given that they can be found in the same file. The said file, is structured such that each line constitutes a rating a user has given to a certain item and follows this configuration:

$$userID, itemID, rating, timeStamp$$

This way, the algorithm just needs to iterate each line and create the respective objects, as the code in Figure 37 depicts.

```
private static UsersRatings LoadUsersRatings()
{
    List<string> lines = GetFileData(RatingsFilePath);

    List<Rating> ratings = new List<Rating>();
    List<User> users = new List<User>();

    int length = lines.Count;
    for (int i = 1; i < length; i++)
    {
        if (lines[i].Trim().Length > 0)
        {
            string[] cols = lines[i].Split(',');

            Rating rating = new Rating(
                Convert.ToInt32(cols[0].Trim()),
                Convert.ToInt32(cols[1].Trim()),
                Convert.ToDouble(cols[2].Trim()),
                Convert.ToInt32(cols[3].Trim()));

            User user = new User(Convert.ToInt32(cols[0].Trim()));

            ratings.Add(rating);
            users.Add(user);
        }
    }


    return new UsersRatings(users.Distinct().ToList(), ratings);
}
```

Figure 37 – Load Users and Items method

After, the tags relative to the items are extracted from the respective file, which contains the following configuration:

$$tagID, name$$

The algorithm only has to iterate each line and create the respective *Tag* object.

Finally, the items are also loaded using a similar approach. In the file where the items are, each line represents an item and follows the configuration presented next:

$$itemID, title, tags$$

The algorithm responsible for extracting the information relative to the items iterates each line and creates an *Item* object for each one, fetching the tags from the already created *Tag* object list.

```
public class DatasetModel
{
    public List<User> Users { get; set; }

    public List<Item> Items { get; set; }

    public List<Rating> Ratings { get; set; }

    public List<Tag> Tags { get; set; }

    public DatasetModel()
    {
        Users = new List<User>();
        Items = new List<Item>();
        Ratings = new List<Rating>();
        Tags = new List<Tag>();
    }

}
```

Figure 38 – *DatasetModel* class

When all the information is extracted from the respective files, the final dataset model can be created, using the *DatasetModel* class, depicted in Figure 38, and with that, the importation and modelling of the data is completed.

6.1.1.2 Class Diagram

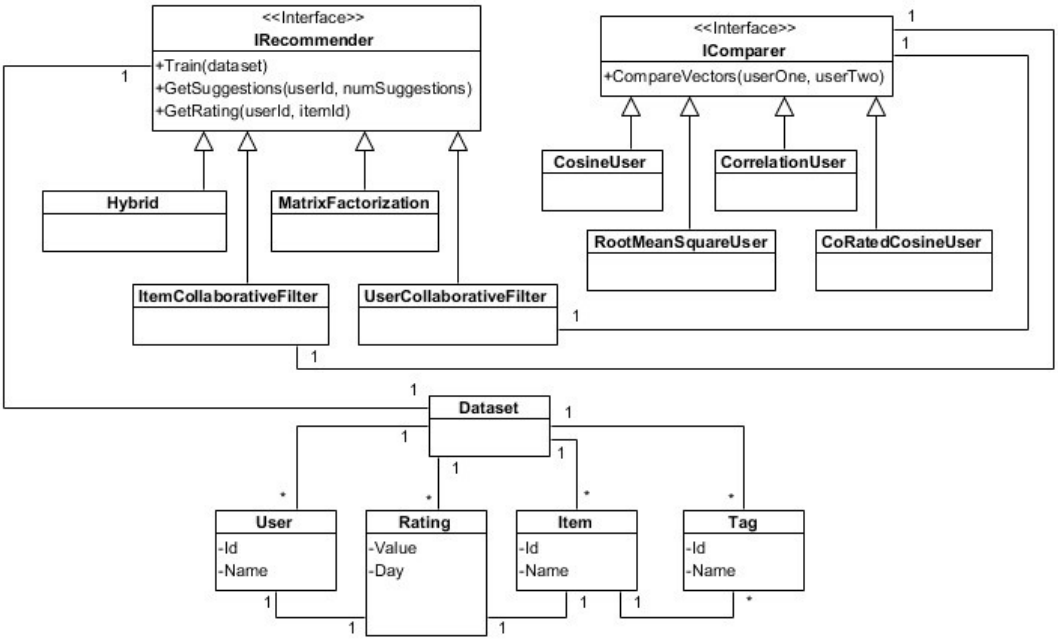

Figure 39 – Class Diagram from the Intelligent Recommender

The previous Figure 39 depicts the class diagram for the system and exhibits its overall structure. The section with the *Dataset* and its dependencies was already covered in the previous section, where the importation and modeling of the data was discussed.

In the next sub-sections, the remaining entities of the class diagram will be explored and furtherly detailed. As an introduction, there are two more "sections" in the diagram, aside from the *Dataset* one, and they are the different recommendation algorithms, all usable through the *IRecommender* interface, and the different implementations for comparers, usable through the *IComparer* interface. These interfaces represent the *Strategy* pattern, which allows for there to be different implementations for the same task and an easy way to make them interchangeable.

6.1.1.3 Matrix Factorization

The core of the devised intelligent recommender can be found on the *MatrixFactorization* class, where this technique is employed. In the State-of-the-art chapter, its theoretical base was described to some extent. Now, in this section, the more intrinsic details will be thoroughly addressed, as well as its full implementation and how it all ties together in the system.

Firstly, the class *MatrixFactorizationRecommender* is instantiated and it can receive as an argument the number of features (*latent features*) that are desired to be used during the learning process. This number of features is simply the size of the user and item features vectors. If no argument is provided, then the default value of 20 is assumed.

The number of *latent features* varies depending on the recommendation model and is mostly defined by trial and error, because if there are too few, then the model won't be able to learn

from the data, and if there are too many, then the model might *overfit*, which is equally bad. The *overfit* problem will be explored later in this section, in a different context.

Regarding the *latent features* number problem, there is some literature that helps to mathematically find the most optimal values, such as using *Cross-Validation* [102] [103]. These techniques won't be investigated in the context of this project, due to time restrictions and intent in building a not over-complicated system in such an early stage. Regardless, it is very much worth exploring as future work.

After that, the model is ready to be trained. The *Train* method receives the previously created *DatasetModel* as input and before anything else, it transforms it into a *user-item ratings* table, or *Rating Matrix* (see Figure 41).

```
public UserItemRatingsTable GetUserItemRatingsTable()
{
    UserItemRatingsTable table = new UserItemRatingsTable();

    table.UserIndexToID = db.Users.OrderBy(x => x.ID).Select(x => x.ID).Distinct().ToList();
    table.ItemIndexToID = db.Items.OrderBy(x => x.ID).Select(x => x.ID).Distinct().ToList();

    foreach (int userId in table.UserIndexToID)
    {
        table.Users.Add(new UserItemRatings(userId, table.ItemIndexToID.Count));
    }

    var userItemRatingGroup = db.Ratings
        .Select(g => new { g.UserID, g.ItemID, Rating = g.RatingValue })
        .ToList();

    foreach (var userAction in userItemRatingGroup)
    {
        int userIndex = table.UserIndexToID.IndexOf(userAction.UserID);
        int itemIndex = table.ItemIndexToID.IndexOf(userAction.ItemID);

        table.Users[userIndex].ItemRatings[itemIndex] = userAction.Rating;
    }

    return table;
}
```

Figure 40 – *Rating Matrix* builder method

In Figure 40, the previously mentioned method responsible for transforming the dataset model is depicted. Firstly, the table object is instantiated – *UserItemRatingsTable* – which initializes its *Users* attribute, that represents a list of *UserItemRatings* (see Figure 42). Each element contains the ID of the user, the list of their ratings and a *Score* attribute, which will not be necessary for this technique specifically. The *UserItemRatingsTable* class also has two more attributes, used to easily convert IDs to Indexes, for users and items.

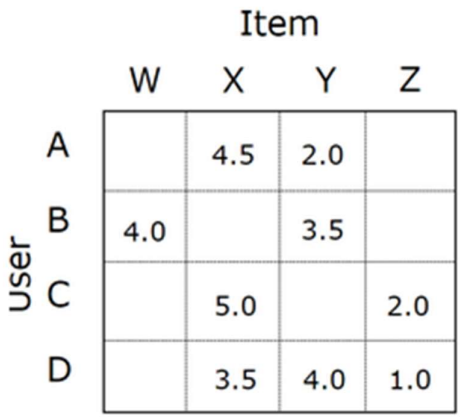

Figure 41 – *Rating Matrix*[29]

After initializing the table, the next step is to populate it. First, the *Users* attribute, containing the list of *UserItemRating* objects is filled, as depicted in the first *foreach* statement of Figure 40. Each element is initialized with the ID of the current user and the number of existent items, which will be the same for all users. The latter will be used to initialize the array of ratings (*ItemRatings* variable) with the proper size, as shown in the next Figure 42.

```
public class UserItemRatings
{
    public int UserID { get; set; }

    public double[] ItemRatings { get; set; }

    public double Score { get; set; }

    public UserItemRatings(int userId, int itemsCount)
    {
        UserID = userId;
        ItemRatings = new double[itemsCount];
    }

}
```

Figure 42 – *UserItemRatings* class

After filling the table with users and items, the table's cells need to be populated with the respective ratings, which is what the *LINQ* and the last *foreach* statement are doing – respectively, creating an object with the ID of the user, the ID of the item and the corresponding rating, and iterating each object, adding the rating to the correct cell. To clarify, the correct cell is given by the Index of the user and the Index of the item, which are the row and column of the table respectively, as illustrated by Figure 41.

[29] Adapted from https://medium.com/@connectwithghosh/simple-matrix-factorization-example-on-the-movielens-dataset-using-pyspark-9b7e3f567536

Having the *Ratings Matrix* created, it is now time to apply the SVD technique. To do so, the *SingularValueDecomposition* class is initialized, receiving as arguments the previously referred number of features and the number of learning iterations, or *epochs*. The latter refers to the iterations that the algorithm will cycle through in order to improve its predicting model. The *epochs* can be seen as a stop rule – at the end of x, the model stops training. There are different rules that can be applied, such as stopping whenever the error is lower than a specific threshold [104], but using learning iterations is the simplest and easiest way.

When a machine learning model tries to make a prediction there's always an error associated, and the ultimate goal is to minimize it as much as possible, through an iterative learning process. The error of a prediction can be easily obtained by computing the difference between the prediction and the real value. This is obviously only applied in training, where the real values exist and can be checked. It is, therefore, a *pseudo-prediction*, because the goal is not to make a real prediction for someone, but to test the overall model, so that going forward one can be confident it will make accurate real predictions.

The idea of evaluating the performance of the machine learning model relates to Loss Function and Cost Function. According to a Towards Data Science article, "[t]he Loss Function computes the error for a single training example while the Cost Function is the average of the Loss Functions for all the training examples." [102] For a set of N data points, each one having a predicted value P' and a real value P, the most simple and common Cost Function, also known as Mean Squared Error (MSE) can be computed as such:

$$Cost = \frac{1}{N}\sum_{i=1}^{N}(P' - P)^2 \tag{4}$$

The goal is to, then, minimize the cost function, because the lower the error, the better the algorithm has done in learning and building a suitable model. And this is where it all ties together with what was described in the State-of-the-art chapter, regarding on how to obtain the correct decomposed matrices to make accurate predictions by recomposing them.

To find the optimal feature matrices, there will have to be some sort of optimization process, to make the associated *Cost Function* as low as possible. To do so, the initial values of these matrices will be completely random. Since there is no real way to obtain their values based solely on non-zero values (as explained in the Matrix Factorization section), the easiest way to have a starting point is to randomize the matrices and go from there. After that, the matrices will be composed, using the dot product of the matrices vectors (user and item features vectors) to compute a prediction and comparing it with the real value, making adjustments depending on the error, and repeating the process for the number of defined learning iterations. These adjustments, or optimizations are done by an algorithm called *Gradient Descent*. There are different variants of this algorithm, as it will be covered later in this section.

As such, after the *SingularValueDecomposition* class is instantiated, the *FactorizeMatrix* method is invoked, receiving as its only parameter the *Ratings Matrix*. This method is where the complete construction of the recommender model will take place, from the randomized initialization of the feature matrices, to the computing of *Loss* and *Cost Functions* for adjusting the matrices to finally obtaining the built model and the associated error.

First, the feature matrices are initialized, with random values, to obtain a starting point, as it was previously described. In the *Initialize* method, the *biases* are also initialized. Those refer to literal biases that users and items might have underlying and that are captured by the model.

Some users might be more inclined to rate items with high ratings, and others might do the opposite, meaning that two users can have the same appreciation for an item and still rate them different – *user bias*. Similarly, different items might have some characteristic that makes users inclined to rate them higher or lower than other ones – *item bias*. Realizing that a specific classification can mean different things is the notion that leads to the inclusion of biases in recommender systems, both of users and items. Including biases for *Matrix Factorization* improves the model and the adjacent recommendations, which makes it very useful. [103] [104] The way these biases work will be detailed later in this section.

After initializing all these variables, the next step in the *FactorizeMatrix* method is to get the average global rating, which will be useful later, and start the iterative learning process. The full algorithm for this can be found in the next Figure 43.

```
double squaredError;
int count;
List<double>  mseAll = new List<double>();

averageGlobalRating = GetAverageRating(ratings);

for (int i = 0; i < learningIterations; i++)
{
    squaredError = 0.0;
    count = 0;

    for (int userIndex = 0; userIndex < numUsers; userIndex++)
    {
        for (int articleIndex = 0; articleIndex < numItems; articleIndex++)
        {
            if (ratings.Users[userIndex].ItemRatings[articleIndex] != 0)
            {
                double predictedRating = averageGlobalRating + userBiases[userIndex] + itemBiases[articleIndex] + Matrix.GetDotProduct(userFeatures[userIndex], itemFeatures[articleIndex]);

                double error = ratings.Users[userIndex].ItemRatings[articleIndex] - predictedRating;

                if (double.IsNaN(predictedRating))
                {
                    throw new Exception("Encountered a non-number while factorizing a matrix! Try decreasing the learning rate.");
                }

                squaredError += Math.Pow(error, 2);
                count++;

                averageGlobalRating += learningRate * (error - regularizationTerm * averageGlobalRating);
                userBiases[userIndex] += learningRate * (error - regularizationTerm * userBiases[userIndex]);
                itemBiases[articleIndex] += learningRate * (error - regularizationTerm * itemBiases[articleIndex]);

                for (int featureIndex = 0; featureIndex < numFeatures; featureIndex++)
                {
                    userFeatures[userIndex][featureIndex] += learningRate * (error * itemFeatures[articleIndex][featureIndex] - regularizationTerm * userFeatures[userIndex][featureIndex]);
                    itemFeatures[articleIndex][featureIndex] += learningRate * (error * userFeatures[userIndex][featureIndex] - regularizationTerm * itemFeatures[articleIndex][featureIndex]);
                }
            }
        }
    }

    squaredError = squaredError / count;
     mseAll.Add(squaredError);

    learningRate *= learningDescent;
}
```

Figure 43 – Iterative learning algorithm from *Matrix Factorization*

As a step-by-step explanation of the previous algorithm, the first *for* loop goes over the number of learning iterations, each iteration having an associated error and an adjustment of the *learning rate* for the next iterations at the end. The *learning rate* is a variable used to adjust the values in the *user* and *item features* matrices, to approximate their values to the real ones. This is a part of the aforementioned *Gradient Descent* optimization algorithm, that seeks to find the minimum value for a given function, in this case, the error function.

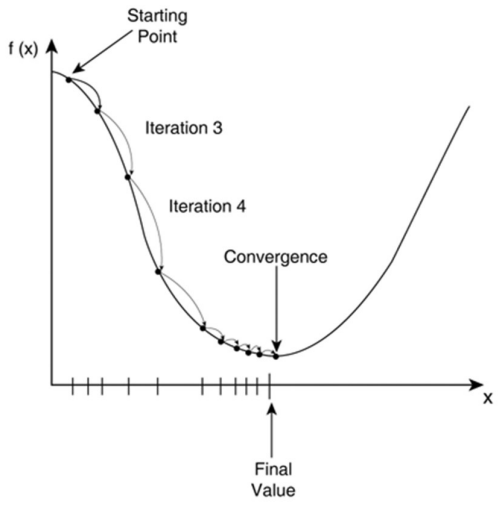

Figure 44 – *Gradient Descent* visualization[30]

Using an example found in a *Towards Data Science* article [105], let's consider one is walking along the path defined by the graph in Figure 44 and is currently at the *Starting Point*. If the goal is to get as close to the *Final Value* as possible, then the only possible course of action would be to go up or down the graph and after taking a step, measure how much far from the goal this new position is and decide again where to go. After taking a step down the graph, the error – difference between the position of the current point and final one – would diminish, validating that one is on the right path. The question now becomes: "how fast must one go?" or "how steep of a step must one take?". If one takes very small steps, then it would take too much iterations to reach the goal (converge), but if the step is too big, then the model might miss the desired minimum. [105] This is where the *learning rate* comes in – as a way to regulate the step that each iteration should take, in order to guarantee convergence. As also illustrated by Figure 44, the step of each iteration is getting smaller and smaller, the closer it gets to the final value, in order to prevent overreaching.

In the devised algorithm, this is evident in the end of each iteration, where the *learning rate* is reduced by 1% each iteration.

Nonetheless, in each iteration, a second *for* loop goes over all the users (rows) and then a third chained *for* loop over all the items (columns). Then, it is checked if the current cell's value – given by the current row and column – is different than zero, because if it is not, then the model can't learn from it, as it was already mentioned several times, and so, it just skips it. If, on the other hand, the cell has a value – if the current user rated the current item – then, the first thing that is computed is the predicted rating, given by the sum of the average global rating, the respective user and item biases, and the dot product between the vectors of the features' matrices. In short, the predicted rating can be given by the dot product only, but the other elements add nuances that improve the computed prediction. For example, if the user tends to rate items very highly, then the predicted rating will be higher too, or if the item tends to be

[30] Image from https://towardsdatascience.com/machine-learning-fundamentals-via-linear-regression-41a5d11f5220

rated very low, then the predicted rating will be lower, or even if the average global rating is high or low, then the predicted rating will be affected accordingly as well. These called nuances make the prediction model more flexible because it takes into account different underlying variables that undeniably play an important role.

After having the predicted rating, then the error can be computed, by calculating the difference between the predicted and the real rating. This error is then squared, to calculate the MSE afterwards.

With the error measured, it is now possible to make adjustments and take a "step" in some direction, optimizing all the variables that are necessary for making predictions – user an item features, average global rating and user and item biases.

These adjustments, or optimizations, are conducted by a *Gradient Descent* algorithm. The variant used for this project is called *Stochastic Gradient Descent* [110] which simply means that the optimization is done individually on each value (cell) at a time, instead of on all the values simultaneously, like some other variants such as the *Batch Gradient Descent* or even *Mini-batch Gradient Descent*. [111]

To update the user and item features, another *for* loop is used, that iterates all the features corresponding to the users' and items' vectors and updates them. According to the literature regarding *Stochastic Gradient Descent and Regularization* [110] and also a *University of Minnesota* course on "*Matrix Factorization and Advanced Techniques*" [108], the formulas to update the step of the user and item features vectors are as such:

$$\Delta q_{if} = \lambda(\epsilon p_{uf} - \gamma q_{if}), \qquad \Delta p_{uf} = \lambda(\epsilon q_{if} - \gamma p_{uf}) \tag{5}$$

Where Δq_{if} and Δp_{uf} are the changes of the values for the item and user features vectors, λ is the *learning rate*, ϵ is the error associated with the current value prediction, and γ is the *regularization term*, which has a static value. The reason why this *regularization term* is being multiplied to the respective feature value and subtracted to the other term is to discourage large user/item features values. This makes it so that there has to be great evidence to support large values. [108] Large values need to be regulated because if they're not, then the learning model might overfit, which is less than desirable. Overfit defeats the entire idea of recommender models, which is to be able to generalize. An example to illustrate what overfitting is can be seen in the next Figure 45.

Supposing we have a dataset, represented by graph number 1 (Figure 45), if we want to build a recommendation model out of the data, and be able to generalize for future instances, then using *Linear Regression* we would get something similar to the graph number 2. If we, on the other hand, try to fit all the data in the model, we might end up with a model similar to graph number 3. This is not a reliable model because it does not capture the dominant trend in the data – which is that is growing – but instead all trends, which makes it near impossible to make predictions on where the next "dot" would be. [109]

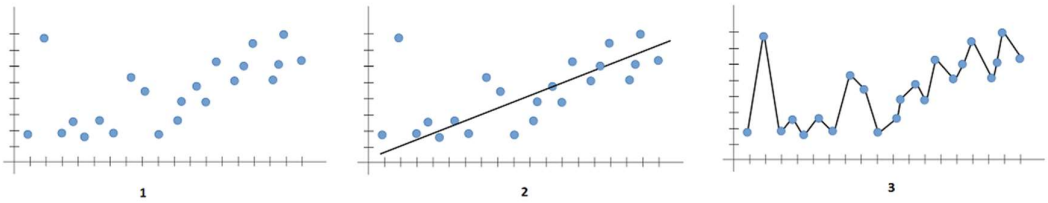

Figure 45 – Overfitting illustration[31]

This is why regularization is important, to prevent large values, *outsiders*, to impact the model negatively by overfitting it. This is supported by literature [107] [110], which offers different types of regularization such as L1, or *Lasso Regression*, and L2, or *Ridge Regression*. These will not be furtherly explored to avoid over-complicating the system at such an early stage but are definitely worth looking into for future work.

Having updated the user and item features vectors for the specific current value, it's still required to update the values of the other variables that impact the prediction formula – average global rating and user and item biases. They all follow the same formula, similar to the ones presented before, regarding the user/item features vectors, with a small adjustment of removing the error multiplier (p_{uf} and q_{if}), where Δv is the changes in the values for the average global rating and user and item biases.

$$\Delta v = \lambda(\epsilon - \gamma v) \quad (6)$$

All these formulas can be found programmatically implemented in the code from Figure 43.

This process is repeated for all the values in the *Ratings Matrix* and at the end of each iteration, the MSE is calculated by dividing the sum of the squared errors for each prediction and dividing for the number of total predictions (number of ratings different than zero). The MSE is then associated to the specific iteration by getting added to a global list. This will be useful to analyze the evolution of the error on each iteration, by drawing a graph that should look like the one in Figure 46.

[31] Adapted from https://towardsdatascience.com/what-are-overfitting-and-underfitting-in-machine-learning-a96b30864690

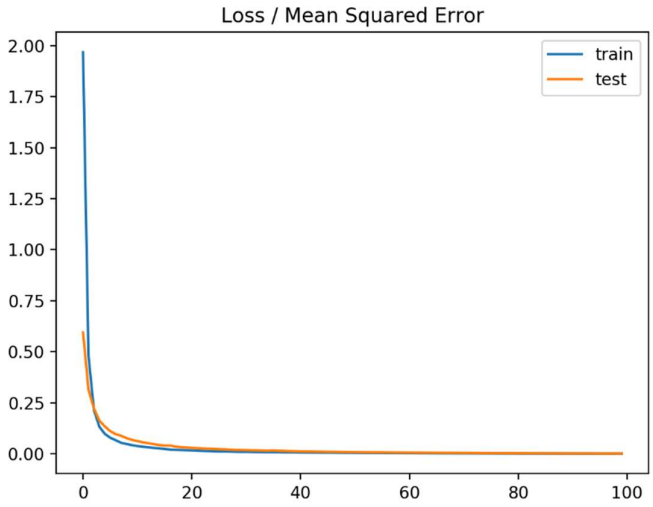

Figure 46 – MSE evolution by iteration

To complete the factorization process, the *SvdResult* object is instantiated and returned, so that suggestions and specific recommendations can be made. This object contains the already named variables needed to make predictions.

Having the prediction model built, *SvdResult*, it is now possible to make recommendations and there are two different types available, that each class under the *IRecommender* interface must implement: get the rating for a specific item by a specific user, *GetRating* method, and get a list of top suggestions for a given user, *GetSuggestions* method.

The *GetRating* method receives the ID of a user and of an item, and using the previously built model, it computes a prediction for how that user would rate that item. That is achieved using the same formula used by *Matrix Factorization* when computing the predicted rating – sum of the dot product of the user and item features vectors, the user and item biases, and the average global rating.

The *GetSuggestions* method is very similar, the only difference being that it applies the *GetRating* method to all the user's unrated items, then sorts them in a descending fashion by their predicted rating and returns the first x items. x being the number of desired suggestions.

6.1.1.4 Comparers

Before presenting and detailing further recommender techniques, it is essential to describe an important method that is on the core of these algorithms. The group of these techniques constitute the *Comparers*. As the name suggests, they make different comparisons and they do it between users and items.

In the State-of-the-art chapter, when discussing *Collaborative Filtering* techniques, a concept that emerged as the main method of application of CF algorithms was the *Nearest Neighborhood* algorithm. This algorithm aims to find the nearest neighbors for a specific

user/item, and to do that, it needs a comparing rule. A user can be similar to another one based on different characteristics, and those different implementations can be found under the *IComparer* interface. There can be, then, multiple implementations for comparers, but the ones included in this project are as follows:

- ***Pearson's Correlation*** [114]: By *Kent State University*'s definition, the *Pearson Correlation* produces a correlation coefficient, "which measures the strength and direction of linear relationships between pairs of continuous variables." [114] The correlation can take on any value in the [-1,1] range. The sign of the value indicates the direction of the relationship, while the magnitude indicates the strength of the said relationship. Being that *-1* is a perfectly negative linear relationship, *0* is no relationship and *+1* is perfectly positive linear relationship. [114] Its formula can be expressed as follows [115]:

$$PCS(a,u) = \frac{\sum_{i=1}^{n}(r_{a,i} - \bar{r}_a) * (r_{u,i} - \bar{r}_u)}{\sqrt{\sum_{i=1}^{n}(r_{a,i} - \bar{r}_a)^2 * (r_{u,i} - \bar{r}_u)^2}} \quad (7)$$

The implementation of said formula is displayed in the next Figure 47.

```
public double CompareVectors(double[] userFeaturesOne, double[] userFeaturesTwo)
{
    double average1 = 0.0;
    double average2 = 0.0;
    int count = 0;

    for (int i = 0; i < userFeaturesOne.Length; i++)
    {
        if (userFeaturesOne[i] != 0 && userFeaturesTwo[i] != 0)
        {
            average1 += userFeaturesOne[i];
            average2 += userFeaturesTwo[i];
            count++;
        }
    }

    average1 /= count;
    average2 /= count;

    double sum = 0.0;
    double squares1 = 0.0;
    double squares2 = 0.0;

    for (int i = 0; i < userFeaturesOne.Length; i++)
    {
        if (userFeaturesOne[i] != 0 && userFeaturesTwo[i] != 0)
        {
            sum += (userFeaturesOne[i] - average1) * (userFeaturesTwo[i] - average2);
            squares1 += Math.Pow(userFeaturesOne[i] - average1, 2);
            squares2 += Math.Pow(userFeaturesTwo[i] - average2, 2);
        }
    }

    return sum / Math.Sqrt(squares1 * squares2);
}
```

Figure 47 – *Pearson's Correlation* implementation

- ***Cosine Similarity*** [115]: Simply put, It is a very popular *collaborative filtering* technique that "measures the cosine angle formed by two rating vectors:" [115]

$$CS(a,u) = \frac{\sum_{i=1}^{n} a_i u_i}{\sqrt{\sum_{i=1}^{n} a_i^2}\sqrt{\sum_{i=1}^{n} u_i^2}} \quad (8)$$

The implementation for this formula can be found in the next Figure 48.

```
public double CompareVectors(double[] userFeaturesOne, double[] userFeaturesTwo)
{
    double sumProduct = 0.0;
    double sumOneSquared = 0.0;
    double sumTwoSquared = 0.0;

    for (int i = 0; i < userFeaturesOne.Length; i++)
    {
        sumProduct += userFeaturesOne[i] * userFeaturesTwo[i];
        sumOneSquared += Math.Pow(userFeaturesOne[i], 2);
        sumTwoSquared += Math.Pow(userFeaturesTwo[i], 2);
    }

    return sumProduct / (Math.Sqrt(sumOneSquared) * Math.Sqrt(sumTwoSquared));
}
```

Figure 48 – *Cosine Similarity* implementation

- ***Co-Rated Cosine Similarity*** [116]: This is a variation of the *Cosine Similarity*, with the same formula. The difference being that only the co-rated items – items rated by both users – are considered and entered in the formula. The code for that is depicted in Figure 49.

```
public double CompareVectors(double[] userFeaturesOne, double[] userFeaturesTwo)
{
    double sumProduct = 0.0;
    double sumOneSquared = 0.0;
    double sumTwoSquared = 0.0;

    for (int i = 0; i < userFeaturesOne.Length; i++)
    {
        if (userFeaturesOne[i] != 0 && userFeaturesTwo[i] != 0)
        {
            sumProduct += userFeaturesOne[i] * userFeaturesTwo[i];
            sumOneSquared += Math.Pow(userFeaturesOne[i], 2);
            sumTwoSquared += Math.Pow(userFeaturesTwo[i], 2);
        }
    }

    if (sumOneSquared > 0 && sumTwoSquared > 0)
    {
        return sumProduct / (Math.Sqrt(sumOneSquared) * Math.Sqrt(sumTwoSquared));
    }
    else
    {
        return double.NegativeInfinity;
    }
}
```

Figure 49 – *Co-rated Cosine Similarity* implementation

- ***Root of Mean Squared Error*** [115]: Scoring rule that measures the average magnitude of the error. This is a technique which gives high weight to large errors, making it very suitable to apply when large errors are particularly undesirable. The formula for that is as follows. [115] [117]

$$RMS(a,u) = \sqrt{\frac{1}{N}\sum(a-u)^2} \tag{9}$$

The programmatic implementation for this technique can be observed in Figure 50.

```
public double CompareVectors(double[] userFeaturesOne, double[] userFeaturesTwo)
{
    double score = 0.0;

    for (int i = 0; i < userFeaturesOne.Length; i++)
    {
        score += Math.Pow(userFeaturesOne[i] - userFeaturesTwo[i], 2);
    }

    return Math.Sqrt(score / userFeaturesOne.Length);
}
```

Figure 50 – *Root of Mean Squared Error* implementation

6.1.1.5 User-Based Collaborative Filtering

The UBCF [118] is a *collaborative filtering* approach that takes into consideration the users and the relationships that might exist between them. These relationships can be calculated using a plethora of techniques, like the ones presented in the previous Comparers section and they can represent different characteristics, such as profiles, purchase patterns or rating patterns, that are inherent to each user.

When initializing this technique's class, two arguments are sent as arguments – the number of neighbors to be used in the *Nearest Neighborhood* technique, and the desired comparing algorithm.

Following its initialization, the way this technique's training is implemented is very simple - Firstly, it is important to mention that it is not an intelligent training, and the term is only used because all recommenders under the *IRecommender* interface must implement the *Train* method. All this method does is get the *Ratings Matrix* (see Figure 40), exactly as in the start of the method for the *Matrix Factorization* technique. After that, the model has everything needed to make suggestions and calculate ratings, there's no need for training of any sort.

The *GetRating* method, presented in Figure 51, receives the ID of the user to which the recommendation is aimed, and the ID of the item to which the model will predict the rating for. Firstly, it finds the user, and their ratings, in the *Ratings Matrix*.

```
public double GetRating(int userId, int itemId)
{
    UserItemRatings user = ratings.Users.FirstOrDefault(x => x.UserID == userId);
    List<UserItemRatings> neighbors = GetNearestNeighbors(user, neighborCount);

    return GetRating(user, neighbors, itemId);
}
```

Figure 51 – *GetRating* parent method for UBCF

Afterwards, the closest neighbors of the said user are computed, using the *GetNearestNeighbors* method, Figure 52 , which receives the user and the number of desired resultant neighbors. What this method does is iterate all the users in the *Ratings Matrix*, compare them individually to the user to which the recommendation is aimed towards, using the specific comparer algorithm sent as an argument in the instantiation of the class, and storing the calculated degree of similarity in a *Score* attribute. At the end, the compared users are ordered in a descending fashion by their *Score* value, and the first x ones will be returned. x being the number of desired resultant neighbors.

```
private List<UserItemRatings> GetNearestNeighbors(UserItemRatings user, int numUsers)
{
    List<UserItemRatings> neighbors = new List<UserItemRatings>();

    for (int i = 0; i < ratings.Users.Count; i++)
    {
        if (ratings.Users[i].UserID == user.UserID)
        {
            ratings.Users[i].Score = double.NegativeInfinity;
        }
        else
        {
            ratings.Users[i].Score = comparer.CompareVectors(ratings.Users[i].ItemRatings, user.ItemRatings);
        }
    }

    var similarUsers = ratings.Users.OrderByDescending(x => x.Score);

    return similarUsers.Take(numUsers).ToList();
}
```

Figure 52 – *GetNearestNeighbors* method

The result from this method, as well as the user and the ID of the item, are sent as an argument to the *GetRating* child method (see Figure 53), which will compute and return the predicted rating for that user and item.

```
private double GetRating(UserItemRatings user, List<UserItemRatings> neighbors, int itemId)
{
    int articleIndex = ratings.ItemIndexToID.IndexOf(itemId);

    var nonZero = user.ItemRatings.Where(x => x != 0);
    double avgUserRating = nonZero.Count() > 0 ? nonZero.Average() : 0.0;

    double score = 0.0;
    int count = 0;
    for (int u = 0; u < neighbors.Count; u++)
    {
        var nonZeroRatings = neighbors[u].ItemRatings.Where(x => x != 0);
        double avgRating = nonZeroRatings.Count() > 0 ? nonZeroRatings.Average() : 0.0;

        if (neighbors[u].ItemRatings[articleIndex] != 0)
        {
            score += neighbors[u].ItemRatings[articleIndex] - avgRating;
            count++;
        }
    }
    if (count > 0)
    {
        score /= count;
        score += avgUserRating;
    }

    return score;
}
```

Figure 53 – *GetRating* child method for UBCF

This method iterates the closest neighbors and for each one it calculates the average rating for all their items. If they rated the intended item, the *score* variable, representing the predicted rating, is updated, adding to it the difference between the rating the current neighbor gave to the desired item and their average rating on all items. This helps to regulate and normalize the predicted rating by taking into account the different rating patterns different users might possess. If a neighbor rated the item very highly that will severely increase the predicted rating, but if the same neighbor tends to rate items highly on average, then maybe the predicted rating should not suffer such a sudden increase.

The next formula (10) is the one used to calculate the predicted rating, where $rating(neighbor_i)$ is the rating the current neighbor gave to the desired item, $average(neighbor_i)$ is the average rating of the current neighbor, n is the number of neighbors that rated the desired item and $average(user)$ is the average rating of the user.

$$rating = \frac{\sum_{i=0}^{n}(rating(neighbor_i) - average(neighbor_i))}{n} + average(user) \quad (10)$$

After iterating all the neighbors, the sum is divided by the number of neighbors that also rated the intended item, and before returning the final prediction, the average rating of the user is added to the value.

```
public List<Suggestion> GetSuggestions(int userId, int numSuggestions)
{
    int userIndex = ratings.UserIndexToID.IndexOf(userId);
    UserItemRatings user = ratings.Users[userIndex];
    List<Suggestion> suggestions = new List<Suggestion>();

    var neighbors = GetNearestNeighbors(user, neighborCount);

    for (int itemIndex = 0; itemIndex < ratings.ItemIndexToID.Count; itemIndex++)
    {
        if (user.ItemRatings[itemIndex] == 0)
        {
            double score = 0.0;
            int count = 0;
            for (int u = 0; u < neighbors.Count; u++)
            {
                if (neighbors[u].ItemRatings[itemIndex] != 0)
                {
                    score += neighbors[u].ItemRatings[itemIndex] - ((u + 1.0) / 100.0);
                    count++;
                }
            }
            if (count > 0)
            {
                score /= count;
            }

            suggestions.Add(new Suggestion(userId, ratings.ItemIndexToID[itemIndex], score));
        }
    }

    suggestions.Sort((c, n) => n.Rating.CompareTo(c.Rating));

    return suggestions.Take(numSuggestions).ToList();
}
```

Figure 54 – *GetSuggestions* method for UBSCF

The *GetSuggestions* method is very similar (see Figure 54). Firstly, it computes the nearest neighbors, in the same fashion as *GetRating* – by invoking the *GetNearestNeighbors* method, Figure 52. It then iterates each item and if the user hasn't rated that them, it will try to compute a prediction for it. The way that is done is by iterating each neighbor, and if they rated the current item, the weighted score for it is calculated and added to the average for that specific item. When all the neighbors are iterated and the next item is ready to be looped, the suggestion for the current item is added to a global list, with its respective score (divided by the number of neighbors). The weighted score is calculated based on the rating the current neighbor gave to the item and also based on a regularization term. This term is a very simple way of saying the further the neighbors list is iterated, the less weight their ratings will have on the final score. Since *GetNearestNeighbors* returns an ordered list of closest neighbors, the first ones will be the most similar, and as such, they should have the biggest weight. The formula for obtaining the score/rating of a specific item is presented next, where $rating(neighbor_i)$ is the rating the current neighbor gave to the current item, n is the number of neighbors that rated

the said item, and $\frac{i+1}{100}$ is the regularization term, which takes a bigger part off of the neighbor's rating the bigger the index of the neighbor is, so that the first neighbor's rating has a much bigger weight than the last's.

$$rating = \frac{\sum_{i=0}^{n}(rating(neighbor_i) \ - \frac{i+1}{100})}{n} \quad (11)$$

After all items are iterated, the suggestions' list is order in a descending fashion, based on the score/rating that it was calculated for each one, and then, the first x suggestions are returned. x being the number of desired suggestions.

6.1.1.6 Item-Based Collaborative Filtering

Another very well-known *collaborative filtering* approach is the IB-CF [118]. This technique, according to a paper published on *Research Gate* "first analyze[s] the user-item matrix to identify relationships between different items, and then use[s] these relationships to indirectly compute recommendations for users". [119] One of the findings suggested by this paper is that "item-based algorithms provide dramatically better performance than user-based algorithms, while at the same time providing better quality than the best available user-based algorithms." [119]

This better performance might have to do with the fact that the calculations associated with user-based approaches take longer to compute, because ideally there are a lot more users than items and items change less frequently than users. [118]

Regardless, the IB-CF works very similarly to the UB-CF, in terms of implementation that is. The only difference being that instead of users, we're comparing the relationships between items. The class is instantiated in the same fashion – providing a comparing algorithm and the number of desired neighbors. Then, the *Ratings Matrix* is fetched, using the method already described and depicted in Figure 40. Afterwards, the matrix is transposed, so that the rows are now the items and the columns the users.

Furthermore, another difference with the UB-CF technique, is the addition of tags to the *Ratings Matrix* which will help to compare items. A *Tag* is a characteristic that each time can possess. In terms of movies, a tag can be its genre, and a specific movie can have multiple genres. Identifying and using tags as means of comparison, aside from the ratings, can be very useful.

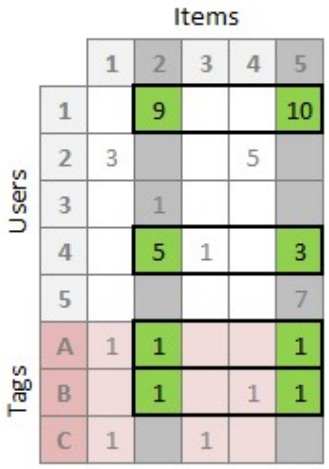

Figure 55 – Transposed *Ratings Matrix* with appended tags[32]

In Figure 55, it is evident the transposed *Rating Matrix* with the appended tags, so that there are more ways to compare items. For example, if we're comparing item *2* and *5*, normally we would find the users who have rated both those items (user *1* and *4*), but now, with the addition of tags, we can also search for items that share the same tags. In the figure's example, both items *2* and *5* share the same *A* and *B* tags, which makes them closely related in terms of similarity.

The model is now ready to make recommendations. As such, the *GetRating* method calculates the desired user's and item's average rating and returns the average of both, as in Figure 56.

```
public double GetRating(int userId, int itemId)
{
    int userIndex = ratings.UserIndexToID.IndexOf(userId);
    int itemIndex = ratings.ItemIndexToID.IndexOf(itemId);

    var userRatings = ratings.Users[userIndex].ItemRatings.Where(x => x != 0);
    var itemRatings = ratings.Users.Select(x => x.ItemRatings[itemIndex]).Where(x => x != 0);

    double averageUser = userRatings.Count() > 0 ? userRatings.Average() : 0;
    double averageItem = itemRatings.Count() > 0 ? itemRatings.Average() : 0;

    return averageItem > 0 && averageUser > 0 ? (averageUser + averageItem) / 2.0 : Math.Max(averageUser, averageItem);
}
```

Figure 56 – *GetRating* method for IBCF

The *GetSuggestions* method, on the other hand, uses a different approach. First, it retrieves the user's top-rated items, using the method in Figure 57. It then iterates this list, and for each one, it finds their nearest neighbors, using the *GetNearestNeighbors* method, almost identical to the one used in the UBCF - Figure 52. Afterwards, it iterates the retrieved list of closest neighbors and computes the average rating of all the users that rated the current item. In the end, returns the first x items with the highest average rating. x being the desired number of suggestions.

[32] Image from https://www.codeproject.com/Articles/1232150/Building-a-Recommendation-Engine-in-Csharp

```
private List<int> GetHighestRatedItemsForUser(int userIndex)
{
    List<Tuple<int, double>> items = new List<Tuple<int, double>>();

    for (int itemIndex = 0; itemIndex < ratings.ItemIndexToID.Count; itemIndex++)
    {
        if (ratings.Users[userIndex].ItemRatings[itemIndex] != 0)
        {
            items.Add(new Tuple<int, double>(itemIndex, ratings.Users[userIndex].ItemRatings[itemIndex]));
        }
    }

    items.Sort((c, n) => n.Item2.CompareTo(c.Item2));

    return items.Select(x => x.Item1).ToList();
}
```

Figure 57 – *GetHighestRatedItemsForUser* method

6.1.1.7 *Hybrid*-approach

As literature suggests [120], and as it was already explored in the State-of-the-art chapter, hybrid recommendation systems have been proposed as a way to improve performance by combining different recommendation approaches. The combination applied in the context of this project is based in the *Matrix Factorization* with different *collaborative filtering* techniques such as *user-based* and *item-based*.

To initialize the *Hybrid* class, the list of desired recommender algorithms to be used is sent and added to the internal list attribute. The training method for this technique is simply iterating the different recommenders training methods. The way recommendations are made, on the other hand, is a bit different.

The *GetRating* method simply invokes each and every *GetRating* method from the used recommenders and averages their results, as in Figure 58.

```
public double GetRating(int userId, int itemId)
{
    return classifiers.Select(classifier => classifier.GetRating(userId, itemId)).Average();
}
```

Figure 58 – *GetRating* method for *Hybrid* filtering

To get suggestions, there are two different alternatives: *GetSuggestions* or *GetCommonSuggestions*.

The *GetSuggestions* method, depicted in Figure 59 first divides the number of desired suggestions equally through all the entered recommender algorithms, so that each one presents its own suggestions. It then invokes each and every *GetSuggestions* method of the recommenders, sorts the results by their rating and returns them.

```
public List<Suggestion> GetSuggestions(int userId, int numSuggestions)
{
    List<Suggestion> suggestions = new List<Suggestion>();
    int numSuggestionsEach = (int)Math.Ceiling((double)numSuggestions / classifiers.Count);

    foreach (IRecommender classifier in classifiers)
    {
        suggestions.AddRange(classifier.GetSuggestions(userId, numSuggestionsEach));
    }

    suggestions.Sort((c, n) => n.Rating.CompareTo(c.Rating));

    return suggestions.Take(numSuggestions).Distinct().ToList();
}
```

Figure 59 – *GetSuggestions* method for *Hybrid* filtering

The *GetCommonSuggestions* method (see Figure 60), on the other hand, as the name suggests, returns the suggestions that are common across all the entered recommender algorithms, as a way of validating their relevance. First, it adds 100 suggestions of each recommender to a list. Then it iterates the said list, creating another list of objects containing the suggestions and the number of their occurrences across the recommenders. In the end, sorts this list by the number of occurrences, to find the most common ones, and then by their ratings. Before returning, it takes the first x suggestions. x being the number of desired final suggestions.

```
public List<Suggestion> GetCommonSuggestions(int userId, int numSuggestions)
{
    int internalSuggestions = 100;

    List<List<Suggestion>> suggestions = new List<List<Suggestion>>();
    foreach (IRecommender classifier in classifiers)
    {
        suggestions.Add(classifier.GetSuggestions(userId, internalSuggestions));
    }

    List<Tuple<Suggestion, int>> final = new List<Tuple<Suggestion, int>>();

    foreach (List<Suggestion> list in suggestions)
    {
        foreach (Suggestion suggestion in list)
        {
            int existingIndex = final.FindIndex(x => x.Item1.ItemID == suggestion.ItemID);

            if (existingIndex >= 0)
            {
                Suggestion highestRated = final[existingIndex].Item1.Rating > suggestion.Rating ? final[existingIndex].Item1 : suggestion;
                final[existingIndex] = new Tuple<Suggestion, int>(highestRated, final[existingIndex].Item2 + 1);
            }
            else
            {
                final.Add(new Tuple<Suggestion, int>(suggestion, 1));
            }
        }
    }

    final = final.OrderByDescending(x => x.Item2).ThenByDescending(x => x.Item1.Rating).ToList();

    return final.Select(x => x.Item1).Take(numSuggestions).ToList();
}
```

Figure 60 – *GetCommonSuggestions* method for *Hybrid* filtering

6.1.1.8 Flow

With all the different techniques covered and detailed, it is now important to describe the system's overall flow.

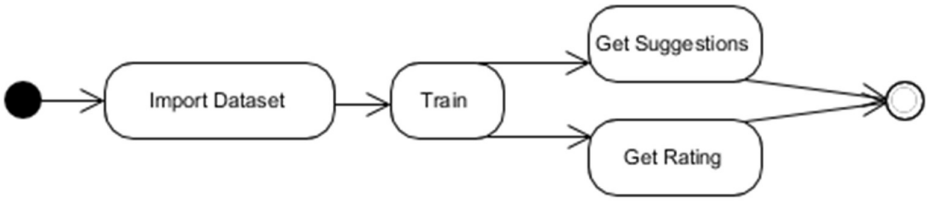

Figure 61 – Activity Diagram for the Intelligent Recommender System

The previous Figure 61 depicts the activity diagram for the Intelligent Recommender System, where after importing the dataset, the model is trained (depending on the technique) and then one can request suggestions for specific users or a predicted rating that a certain user would give to a particular item.

Figure 62 exhibits a more detailed description of the flow of the system, where firstly the desired recommendation technique is chosen. It usually is *Matrix Factorization* or a *Hybrid* combination with other one(s). After that, it is verified if there is already a model for the selected recommendation technique. If there is, then it's ready to make recommendations. If there is not, then the model is created, using the steps already covered in the previous sections. This model is then stored for future use.

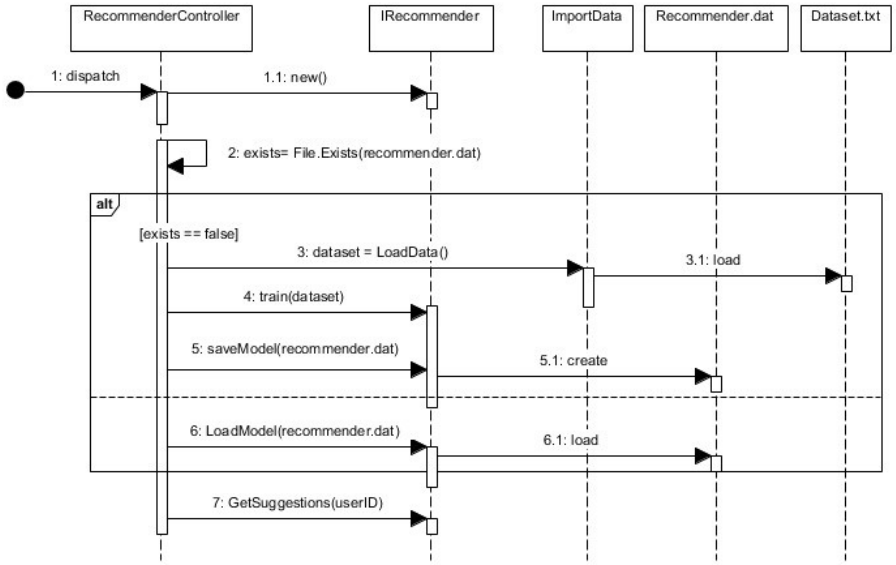

Figure 62 – Sequence Diagram for the Intelligent Recommender System

6.1.2 Conditional Recommender

Having implemented an intelligent recommender based on machine learning and iterative improvement processes, there was still the need to possess a smaller, faster, and more direct recommender system. Even though smart recommendations are better in most cases, they require a very specific set of conditions to work properly. The first and most important one being the great amount of initial testing data.

Unless there's already a considerable volume of information ready to feed the system to build the recommendation model, then the recommendations will be far from accurate and probably more on the random side. This is exactly the situation with this project's system – even though the algorithm was properly built to support the envisioned model and will be trained and tested with similar data (in terms of structure, not context), it ultimately cannot provide context-true results until applicable data is gathered.

For this reason, an additional recommender will be implemented, to offer fast and simple recommendations in the absence of more robust ones calculated from the intelligent recommender. The end recommendation object is in the form of training Workouts. It won't recommend training Plans or Programs because even though they have the same logic, due to time restrictions only one could be selected.

6.1.2.1 Theoretical Base

Even though the envisioned algorithm is desired to be fast and simple, it still has some intricacies on its workings that deserve to be presented.

Firstly, the algorithm will not work so much as a "workout builder" but more as a "workout filter". That is to say that it won't build workouts from zero and recommend them - this would be highly difficult because it would require a more scientific and research-based knowledge in workout programming and defeat the whole simple-and-fast purpose. Instead, it will use the already existing workout database, and by filtering process, select the most appropriate ones, depending on the context (inputs).

To do so, it will receive the as input the following parameters:

- ***Workout Database***: The workout list that will be filtered.
- ***User History***: User's workout history, to find tendencies, lacking areas, etc.
- ***User Preferences***: Preferences of the user, established during the setup – available equipment, muscle groups, training types, body areas, goals and level of experience.

To sum up, the inputs comprise the workout database, which is the list of existing workouts in the system's database, the user's workout history, which will help to find what the particular user values most, find tendencies in their training, areas that need more work, etc. It also comprises the user preferences, which were defined during the setup, and include the equipment the user has access to, muscle groups that the they want to work, the training types they want to practice, the body areas they want to hit, their training goals and their level of experience.

Regarding the filtering technique more specifically, the idea is to divide it in 6 stages – filters refinement, volume and intensity evaluation, filtering, volume and intensity prediction, final selection.

The first stage, *filters refinement*, is the simplest one. Its main purpose is to take the user preferences and cross-reference them with the user's history to refine them. For example, if the user's training types were already all included in their previous workouts but one, then it makes the most sense to recommend a workout of that specific training type. And so, the list of training types would be reduced to just the one.

The second stage, *volume and intensity evaluation*, has to do with assessing the volume and intensity the user has been subjected to in their previous workouts, to later decide what would constitute the most adequate volume and intensity for the next workout and find one that is the most alike.

The third stage, *filtering*, is the phase where the refined parameters of the first stage will be applied in the workout list, reducing the number of workouts to ones that theoretically will be of most interest to the user. First, the workouts that are above the user's experience level will be removed from the list. Then, only the workouts whose goals, training types and muscle groups align with the previously refined goals, training types and muscle groups respectively will be kept. Also, the workouts which use exercises that require equipment the user has no access to are to be removed as well.

The fourth stage, *volume and intensity prediction*, takes into consideration the evaluation conducted in stage two, and tries to make a prediction on what would be most suitable to the user in terms of volume and intensity. From this point forward is where the results can achieve a greater degree of differentiation, depending on what is fed to the stage's algorithms. This is because there are numerous ways to predict volume and intensity and to make a final selection.

Even though the other stages can achieve a certain degree of differentiation, for example, there are several ways to evaluate volume and intensity, they won't achieve the same degree that the fourth and fifth stage will.

The implemented prediction of volume an intensity is based on the assumption that the detected tendency of past workouts will remain. To calculate this tendency, *Linear Regression* will be used. This technique is one of the most important ones in *Regression Analysis*, and its main purpose is to establish a relationship between a dependent variable and one or more independent variables using a line, also known as *Line of Best Fit*, or *Regression Line*. [99]

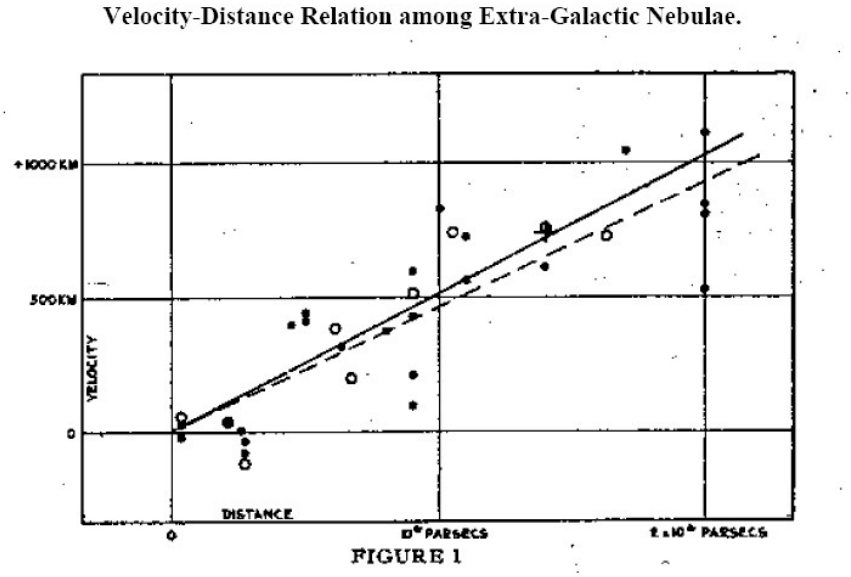

Figure 63 – *Hubble's Constant*, expressed through the slope of a *Linear Regression* graph[33]

In the previous Figure 63, the depiction of a standard *Linear Regression* graph is evident. Even though the data in the independent variable (Y-axis) is scattered, there's a tendency upwards, more clearly noticeable through the use of the line – *Line of Best Fit*. This line indicates that, regardless of *outliers*, there's an unmistakable growth, represented by the slope of the line. With this, one might predict what the *velocity* (Y-axis) might be for a specific unmeasured *distance* (X-axis).

Accordingly, applying the same technique to the volume and intensity of the user's past workouts will result in a line that may represent a tendency that the user subconsciously wants to maintain, and with such line, a prediction can be easily made.

Still, there are some important points that need to be established. Firstly, it is assumed that there's a *linear relationship* between the variables [99]. In other words, it is assumed that the user's volume an intensity evolution is related to the evolution from workout to workout, which may not be true for some cases. Secondly, this technique is very sensitive to *outliers*, especially when data is scarce. This must be payed attention to, with the consequence of making unrealistic predictions.

In the fifth and final stage, *final selection*, the predicted values for the volume and intensity are used to filter the list of workouts and obtain the ones that best fit the predictions, depending on how many are expected to be returned.

[33] Image from https://astro.unl.edu/naap/distance/hubbles_law.html

6.1.2.2 Flow

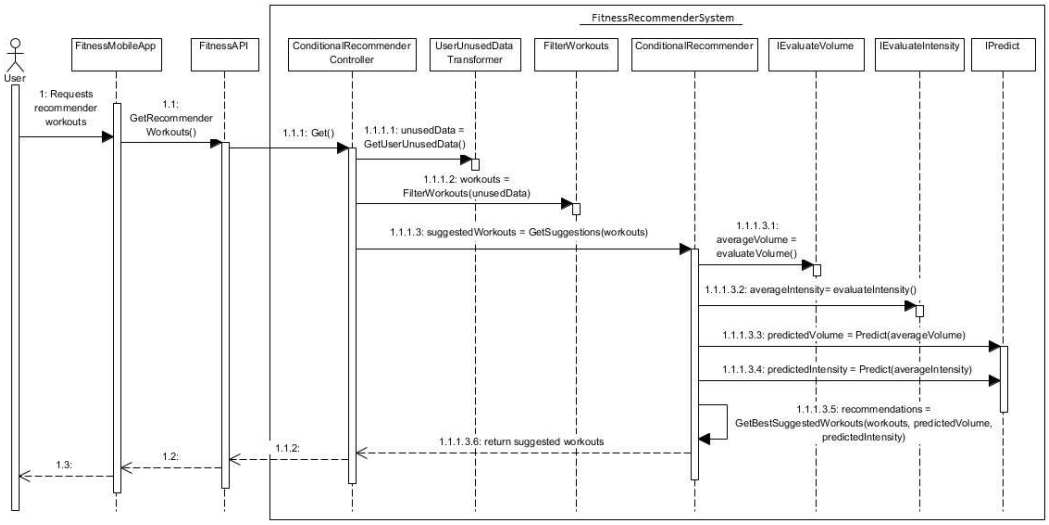

Figure 64 – Sequence Diagram: Conditional Recommendation

In the previous Figure 64 the sequence diagram relative to the conditional recommender process' flow is presented. The process involves three entities: *Mobile APP*, *API* and *Recommender System*.

The first entity, represented by the *FitnessMobileApp* component, gets the input from the user that a recommendation is requested. Then, the server, *FitnessAPI*, is communicated with, that in its stead communicates with the recommender system, *FitnessRecommenderSystem*, sending the user's info, and other inputs, already presented in the previous section. Finally, the recommender processes the gathered information and returns it back in the form of workout recommendations. This process was also already described in the previous section.

6.1.2.3 Implementation

This final section will be dedicated to detailing and describing the specific implementation of the recommender system's filtering process, which was already fully described previously.

```
public List<General> getUserUnusedGoals(List<SingleWorkout> userHistory, List<General> goals)
{
    List<General> goalsHistory = userHistory.Select(x => x.Goals).SelectMany(i => i).Distinct().ToList();

    return goals.Where(x => !goalsHistory.Any(y => y.ID == x.ID)).ToList();
}
```

Figure 65 – Method to get the unused goals

As it was already discussed, in the *filters refinement* stage, the user's preferences are cross-referenced with their history to find "lacking areas". This is exactly what is being portrayed in Figure 65, where the user's goals are being refined. More specifically, the method is searching for goals not present in the workout history and returning them, completing the refinement process for that preference. Each workout has a set of goals, and the algorithm is simply trying

to find the ones that were not "fulfilled" in the user's history (list of workouts), which would make workouts that have those goals suitable candidates to recommend. The other preferences follow the same method of refinement.

Regarding the *volume and intensity evaluation* stage, the *Strategy* [100] pattern was employed, so that the code would be reusable, and to allow different evaluation algorithms to be selected, independently of who, or what, is using them, as shown in Figure 66, where *RepsEvaluate* and *PercentageRpeEvaluate* implement their respective interface. The idea here is that other classes, with different approaches, can implement the same interface, reusing the code.

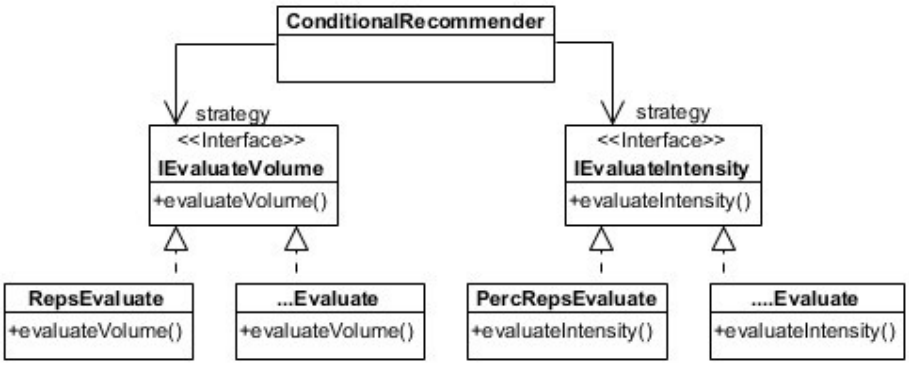

Figure 66 – *Strategy* Pattern: *IEvaluate*

To accomplish what was set on this stage, two classes were created – *RepsEvaluate* and *PercentageRpeEvaluate*. The first implements the *evaluateVolume* method that calculates the volume of workouts, which is based on the number of repetitions for each one, returning a list of *integers*, each representing the volume of a workout. The former implements the *evaluateIntensity* method, that calculates the intensity of each workout using a different approach.

As it was described in previous chapters, specifically in the Domain Model, each workout contains a list of exercises, each exercise contains a list of sets, and each one has an attribute referring to its intensity. The main idea behind the algorithm from *PercentageRpeEvaluate* is to associate each exercise with an intensity, calculating the average between its sets, and then averaging the exercises' intensity to obtain the intensity of the workout, as the following formulas (12) portray, from left to right respectively.

$$I_{exercise} = \frac{I_{set_1} + I_{set_2} + \cdots + I_{set_n}}{n}, \qquad I_{workout} = \frac{I_{exercise_1} + I_{exercise_2} + \cdots + I_{exercise_n}}{n} \tag{12}$$

In the *filtering* stage, the refined parameters from the first stage are used to filter the list of workouts, as shown in Figure 67.

```
private List<SingleWorkoutDTO> filterGoal(
    List<SingleWorkoutDTO> recommendations, List<GeneralDTO> unusedGoals, List<GeneralDTO> userGoals)
{
    List<GeneralDTO> goals = unusedGoals;
    if (goals.Count == 0)
    {
        goals = userGoals;
    }

    var temp_list = recommendations.Where(x => x.Goals.Any(y => goals.Any(z => z.ID == y.ID))).ToList();
    if (temp_list.Count > 0)
        return temp_list;
    else
        return recommendations;
}
```

Figure 67 – Method to filter the workout list by their goals

The previous method receives as arguments the list of current *recommendations* (workout list) to filter it, the *unusedGoals*, which is the refined goals from the first stage, and the un-refined goals list. The reason for the latter is to have a "backup" in case the refined goals list is empty. If it is, then the refined list becomes the original one. After that validation, the workout list is filtered, to find which ones contain at least one goal from the said list. If the resultant list is empty, then the "pre-filtered" one is returned.

The previous algorithm is the same one applied to all the other preferences. Also, the workouts which have exercises that require equipment not included in the user's available equipment list, are filtered out.

The *volume and intensity prediction* stage uses, once again, the *Strategy* pattern, to allow for there to be different implementations for the prediction of volume and intensity. The implemented one though, is based on the already thoroughly described technique – *Linear Regression*.

With this technique, the *Line of Best Fit* can be calculated, and in order to produce predictions, it needs to be translated into an equation, the straight-line equation, in the "slope-intercept" form [101]:

$$y = mx + b$$

where m is the slope and b gives the y-intercept

To obtain this equation, three steps need to be taken. First, the average of the x (workout index) and y (volume/intensity) is calculated, $\bar{x}$ and $\bar{y}$ respectively. Then, the slope, m, is obtained through the following formula (13).

$$m = \frac{\sum_{i=1}^{n} x_i y_i}{\frac{n - \bar{x}\bar{y}}{\sum_{i=1}^{n} {x_i}^2}} \quad (13)$$
$$\frac{}{n - \bar{x}\bar{x}}$$

Finally, the y-intercept value, b, is calculated, using the formula presented next.

$$b = m\bar{x} - \bar{y}$$

These steps are programmatically implemented in the code shown in Figure 68.

```
public static List<XYPoint> GenerateLinearBestFit(List<XYPoint> points)
{
    int numPoints = points.Count;
    double meanX = points.Average(point => point.X);
    double meanY = points.Average(point => point.Y);

    double sumXSquared = points.Sum(point => point.X * point.X);
    double sumXY = points.Sum(point => point.X * point.Y);

    double a = (sumXY / numPoints - meanX * meanY) / (sumXSquared / numPoints - meanX * meanX);
    double b = (a * meanX - meanY);
    [...]
}
```

Figure 68 – *Line of Best Fit* equation implementation

With this equation already available, predicting values for volume and intensity becomes very easy. It is only needed to replace x with the index of the next workout, and return the respective y value, that represents the said prediction.

The last stage, *final selection*, the predictions gathered previously will be used to make a final filtering on the workout list, that selects the first n workouts that best approximate the predictions made for the volume and intensity. Given that n is the number of recommendations that will be presented to the user, which can be changed. After the final workout list is returned, the conditional recommender process is finished.

6.2 Mobile Application

This section will be dedicated to the presentation of the developed mobile application. First and foremost, it can be said that it was built using *React Native*[34] as its main framework, *Expo*[35] as a tool for building, deploying and iterating the app, and *Visual Studio Code*[36] as the code editor.

React Native is a JavaScript[37] framework, used to build native mobile applications for both iOS and Android. Based on the Facebook's JavaScript library – *React*[38] – but instead of the web and browser, it targets mobile platforms. [125] There are numerous advantages to using *React Native*, the first and most important is the fact that it supports different mobile platforms natively, removing the need to develop for both – the code will work for both. Also, it uses a very-well known programming language, JavaScript, which makes familiarization with the technology an incredibly easy process.

[34] https://facebook.github.io/react-native/
[35] https://expo.io/
[36] https://code.visualstudio.com/
[37] https://www.javascript.com/
[38] https://reactjs.org/

This framework is being used by thousands of applications, the most well-known being Facebook, Instagram, Pinterest, Skype, Tesla, Uber, and many more mobile based applications.[39]

The other technology used, *Expo*, is so notorious and relevant that it is even referenced in the "*Getting Started*" [40] page of the *React Native* docs. It is said that the easiest way to get started is with *Expo CLI*[41], which is a command line app that serves as the main interface between a developer and other *Expo* tools. As such, *Expo* can be seen as an aggregate of different tools aimed towards the expedition of mobile apps with the less amount of effort. It's very useful because it can be tested on an iOS device without paying the annual *Apple Developer Account* fee of $99[42]. This is attainable through the *Expo* app, both in the Google Play and App Store, which emulates the app that is being developed in the desired device. Furthermore, *Expo* handles tons of configurations instead of the developer and even has an SDK that allows the app to use the device's camera, maps, location, etc. [126] Another tool from the *Expo* package is the *Snack*[43] which is a handy online editor, that allows to run the application on an online emulator or even on the developer's own device.

The last used technology to build the mobile application was *Visual Studio Code*, which is a "lightweight but powerful source code editor […]. It comes with built-in support for JavaScript, […] and has a rich ecosystem of extensions for other languages […]." [127]

The next sub-sections will unveil specific details regarding the developed system's characteristics, such as the used patterns and supported features, the libraries used, and at the, end the most important flows of execution of the User Interface will be presented and detailed.

6.2.1 System's Characteristics

The developed mobile application exhibits a plethora of characteristics that are worth noting and documenting. These constitute decisions made in terms of used patterns, available features and even simply overall characteristics with a certain degree of importance to the proper functioning of the system.

6.2.1.1 Redux

Simply put, *Redux* [128] is a state container for JavaScript apps. It helps to build applications that are predictable, centralized, debuggable and flexible.

Predictable because they behave consistently, run in different environments and are easily testable. Centralized because the application's state and logic is centralized, enabling different state handling capabilities. Debuggable because using it makes it easy to trace the application's state. And flexible because it works with any UI layer and has a large ecosystem of addons.

---

[39] Data from https://facebook.github.io/react-native/showcase.html
[40] https://facebook.github.io/react-native/docs/getting-started.html
[41] https://docs.expo.io/versions/latest/workflow/expo-cli/
[42] Data from https://developer.apple.com/support/compare-memberships/
[43] https://snack.expo.io/

Even though it is mostly used with *React*, it can be used with any other JavaScript framework or library. [129]

The reason why state management is important is because in a mobile application, data is usually shared among different components, and there has to be some sort of control over the data's state that is being shared, to avoid any mistakes. [129]

It is now important to understand how *Redux* works in terms of implementation [129]. It's not very complicated and rather straight-forward. Firstly, there are three important parts in any *Redux* system: *actions*, *store*, *reducers*.

The *actions* [130] are events, they're the way one can send information from the application to the *Redux* store, described later. These *actions* are sent using a *store.dispatch()* method and they must possess a type property indicating the type of action that is going to be carried out and a payload, which is the information that is desired to be stored, as in Figure 69.

```
{
  type: ACTION_TYPE,
  payload: 'Hello World!'
}
```

Figure 69 – *Redux action* example

The *store* [131] simply put, holds the application's state and the only way to change it is to dispatch an *action* on it. The *store* is not a class, but rather an object holding multiple methods, such as *getState()*, which returns the current state of the application and *dispatch(action)*, which dispatches a specific action to trigger a state change. There are others, but the ones used in the project, and the most important ones are the ones described.

The *reducers* [131] are functions that take the current state of the application, perform some action and return a new state, as in Figure 70. They "specify how the application's state changes in response to *actions* sent to the store." [131]

```
function example(state = initialState, action) {
  switch (action.type) {
    case ACTION_TYPE:
      return Object.assign({}, state, {
        payload: action.payload
      })
    default:
      return state
  }
}
```

Figure 70 – *Redux reducer* function example

To connect a specific screen with the *Redux* store, the *connect()* method [133] is used. This function connects a React component to the *Redux* store, and it receives two useful arguments – *mapStateToProps()* and *mapDispatchToProps()*. The first serves almost as a *subscriber* to the *Redux*'s store, where any change to it will trigger this function and be updated. If there's no

need to subscribe to any specific variable in the store, the *null* value can be passed. The latter, on the other hand, is used to create functions that dispatch actions, receiving objects that the screen can use. [133] All of this can be seen in Figure 71.

```
const mapStateToProps = state => {
    return {
        equipment: state.availableEquipment.equipment,
        language: state.language.currentLanguage,
    };
};

const mapDispatchToProps = dispatch => {
    return {
        onGetEquipmentList: () => dispatch(getEquipment()),
        onGetLanguage: () => dispatch(getLanguage()),
    };
};

export default connect(mapStateToProps, mapDispatchToProps)(HomeScreen);
```

Figure 71 – Redux connect example

6.2.1.2 Async Storage

The *AsyncStorage* [133] "is a simple, unencrypted, asynchronous, persistent, key-value storage system that is global to the app." [133] In the system, is very useful because, unlike *Redux*, if the application is closed, the data will still be accessible, due to being stored in a sort of local "cache" on the device. On the other hand, *Redux*'s data is initialized whenever the app is re-opened. An example of how *AsyncStorage* works can be found on Figure 72, where an example of storing data is illustrated.

```
_storeData = async () => {
  try {
    await AsyncStorage.setItem('@MySuperStore:key', 'I like to save it.');
  } catch (error) {
    // Error saving data
  }
};
```

Figure 72 – *AsyncStorage* persisting data example[44]

[44] Adapted from https://facebook.github.io/react-native/docs/asyncstorage

Contrarily, an example of fetching the same data can be seen in Figure 73.

```
_retrieveData = async () => {
  try {
    const value = await AsyncStorage.getItem('TASKS');
    if (value !== null) {
      // We have data!!
      console.log(value);
    }
  } catch (error) {
    // Error retrieving data
  }
};
```

Figure 73 – *AsyncStorage* fetching data example[45]

For the context of this specific project, there are three important *AsyncStorage* variables that are worth mentioning. These are the *language*, *height unit* and *weight unit*. These are useful to be saved in this manner because the fact that the app can be closed and re-opened and the variables will still hold their value, which means that there's no need to fetch these configurations again from the server.

Regardless, there are various other variables, that will not be discussed further, due to the fact that are not relevant for this project, but for the other author's (see Methodology).

6.2.1.3 Components

React allows for the definition of different components as function components or even class components. This way, these components can receive different arguments, accessible to the *props* property and they return either a function or an entire class. The latter refers to UI classes, with their own *render* method.

This encourages component reutilization, given that it only has to be defined once, and then it can be customized by the entity invoking it. For example, if a text class component is defined, which renders a text object and receives *props* such as its style, then, different invokers can specify different styles for the same text component, reusing it.

The project employed this technique, seen as a good design pattern, as a means to reduce the sheer amount of existent code. The fact that if some changes on a specific component are desired, they can be applied to the reusable component, and all its instances would be changed as well constitutes a huge advantage in terms of maintainability.

[45] Adapted from https://facebook.github.io/react-native/docs/asyncstorage

6.2.1.4 Navigation

Mobile apps are usually characterized by possessing multiple screens, and as such, there has to be some sort of manager that handles the presentation, transition and declaration of those screens. That manager is called a *navigator*. [134]

In the context of this project, that manager is *React Navigation*, which "provides an easy to use navigation solution, with the ability to present common stack navigation and tabbed navigation patterns on both iOS and Android." [134]

There are different types of navigators, the used ones are as follows:

- ***Tab Navigator*** [135]: Bottom or top tab that allows for easy swap of screens.
- ***Drawer Navigator*** [136]: Lateral drawer menu.
- ***Stack Navigator*** [137]: Provides a way to transition between screens where each one is placed on top of a stack.
- ***Switch Navigator*** [138]: Its purpose is to only ever show one screen at a time.

These described navigators can all be found in the following Figure 74, where the relationship between them and their respective screens is illustrated.

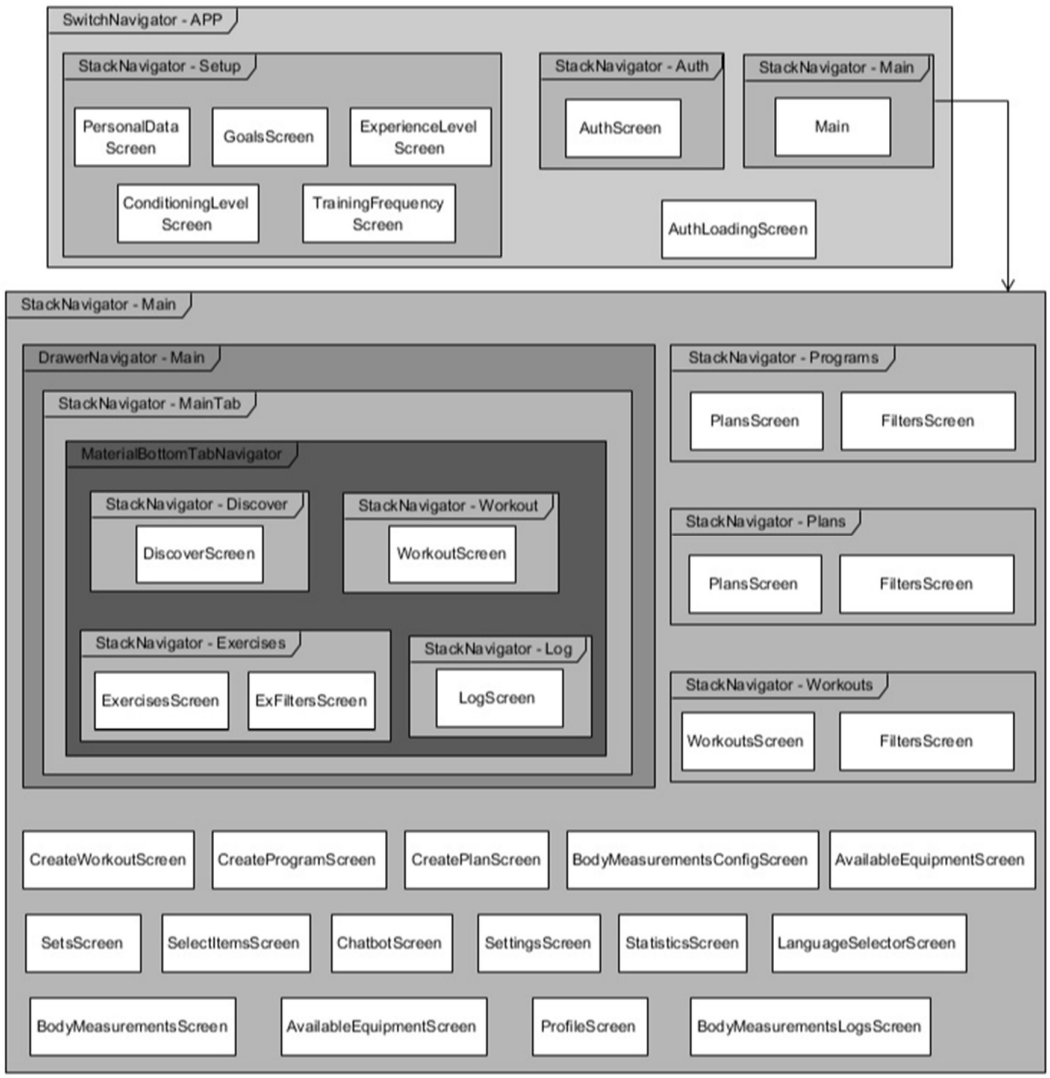

Figure 74 – Mobile App's navigators and respective screens

6.2.1.5 Authentication

One of the defined *Non-Functional Requirements* had to do with security through *Authentication*. To achieve this, it was implemented an authentication system on the mobile application where it's demanded for a registered account to be able to access its features. If the user doesn't have an account, they can create one. They can also use the "forget password" feature, where they receive an e-mail confirming its identity and then a new password is generated.

6.2.1.6 Multi-language

One of the most fundamental aspects of the mobile application is the fact that supports multiple languages and offers the addition of new languages with minimal effort, by the way it was implemented. The used process is named *Internationalization*, or *I18N* for short, and by definition "is the process of planning and implementing products and services so that they can easily be adapted to specific local languages […]." [134]

To implement this said process, the *i18n-js* [46] library was used, which offers a simple way of implementing *internationalization* in *JavaScript* apps. To use it, 3 elements were created: *index*, *language list*, *language file*.

The *index* is where some configurations are made, regarding the importation of *language files*, default language, fallbacks (for example if one wants "en-US" and "en-GB" to fallback to "en"), among others.

The *language list* is the file containing the different available languages, and their *locale* key, as shown in the code snippet from Figure 75.

```
const languages = [
    {
        locale: 'en-US',
        name: 'English'
      },
      {
        locale: 'pt-PT',
        name: 'Português',
        englishName: 'Portuguese'
    }
]

export default languages;
```

Figure 75 – *LanguageList* file example

The *language file* is the file where all the translations for the different variables are. There are as many *language files* as there are available languages. In Figure 76, an example of how a *language file* would look is depicted.

[46] http://i18njs.com/

```
{
  "values":{
    "Yes": "はい",
    "No": "いいえ",
    "Ok": "Ok",
    "Cancel": "キャンセル"
  }
}
```

Figure 76 – *LanguageFile* example

The usage of this library is very straight-forward – it's just needed to invoke the *i18n.t()* method, passing as parameter the key of the *language file*'s desired variable. For instance, in the Figure 76 example, if it is desired to get the translation for the "cancel" variable, the method would look like *i18n.t(values.Cancel)*, which would return the respective translation.

6.2.2 Libraries

Many functionalities regarding UI elements were achieved using already built libraries, available on *GitHub*[47]. The next table Table 7 depicts all the used libraries, as well as the number of *Stars* they have on *GitHub*. These refer to their popularity on the site.

Table 7 – Library list

Library Name	**Description**	**Stars**[48]
react-native-animatable [135]	Animates different components.	7103
react-native-app-intro-slider [136]	Sliders/Swipers.	800
react-native-calendars [137]	Calendars.	1319
react-native-collapsible [135]	Animated collapsible component.	1572
react-native-dialog [139]	Native dialog/alert.	209
react-native-elements [140]	UI Toolkit.	17135
react-native-gifted-chat [141]	Chat UI.	7981
react-native-loading-spinner-overlay [142]	Spinner UI overlay.	1146
react-native-material-menu [143]	Material menu component.	223
react-native-material-textfield [144]	Material textfield.	630

[47] https://github.com/
[48] All data from 25/9/19

react-native-modal [145]	Modal UI component.	2846
react-native-paper [146]	UI Toolkit.	4208
react-native-parallax-scroll-view [147]	Animated parallax header.	1814
react-native-picker [148]	Native wheel picker.	1519
react-native-picker-select [148]	Native picker.	575
react-native-progress [149]	Progress indicators and spinners.	2514
react-native-pure-chart [150]	Chart library components.	198
react-native-really-awesome-button [151]	Buttons toolkit.	791
react-native-search-box [152]	Animated search bar.	370
react-native-sectioned-multi-select [153]	Multi-selectable modal.	347
react-native-snap-carousel [154]	Swiper component.	5954
react-native-sortable-listview [155]	Drag and drop list.	839
react-native-stopwatch-timer [156]	Stopwatch and timer component.	45
react-native-swipeable [157]	Swipeable list item.	842

6.2.3 User Interface

Having presented the mobile application's characteristics, design patterns, used libraries and even the employed technologies, it is now important to detail the main execution flows in terms of user interface. These will serve as an introduction to the more specific detailing of screens and interfaces in the next Server Application section.

The first screen a new user would encounter would be one depicted in Figure 77. There, the user is firstly faced with an introductory screen with the main features of the app such as the recommendations it provides, the personal virtual assistant, and others. After that, the login screen is shown, where the user can login, create a new account, or reset their password. After login in for the first time, the setup screens appear, in order to define the user's profile, collecting information regarding their age, goals, workout frequency, and others. Finally, the user is finally inside the app itself, displaying the discover screen, where the user can perform a plethora of actions.

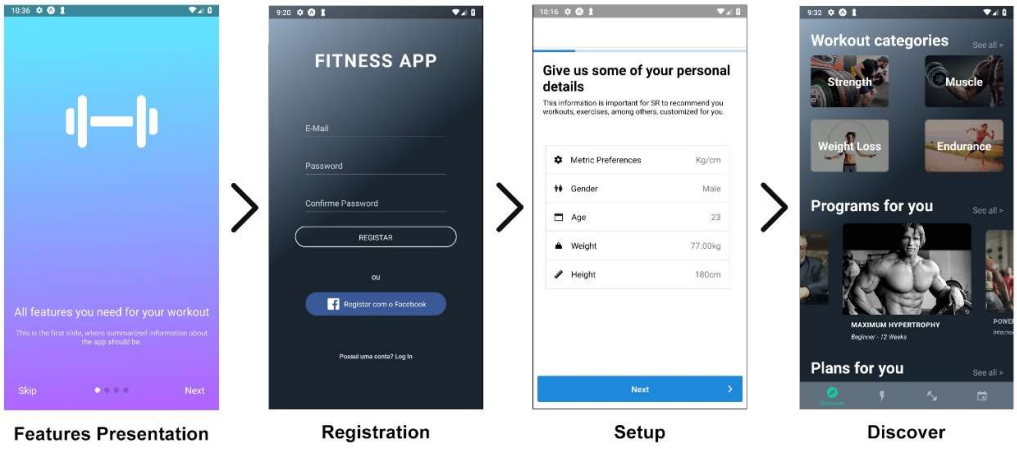

Figure 77 – Mobile App UI: Initial Flow

One of the few actions that can be performed from the discover screen is the visualization of training programs, plans and workouts. From a different tab other than the "discover" one it's also possible to view the exercises. The actions can all be found in Figure 78.

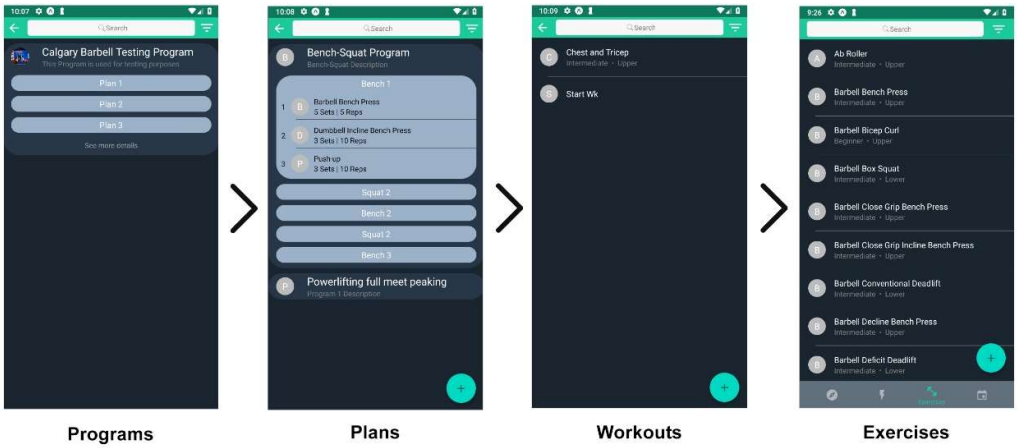

Figure 78 – Mobile App UI: Training Lists

The user might also want to create and run their own training workouts, which is what is being displayed in Figure 79, where firstly a workout is being created and then it's being started and ran in real time. Finally, the user might also desire to consult their logs, referring to past or future programmed workouts.

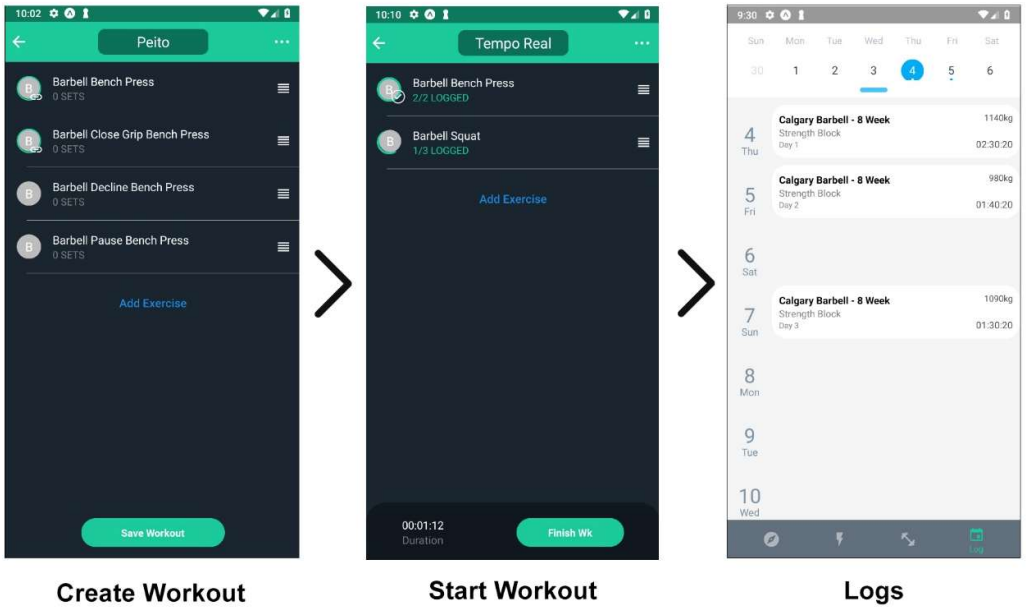

Figure 79 – Mobile App UI: Create, Start and Check Workouts

Another very important flow that new users might find helpful is the configuration of the app to their personal taste. To do that, there's a side drawer menu where different screens can be accessed. One of those screens is the settings one, where the user can change things like the default units of measure, system's language and the equipment they have access to. In the profile screen, they can also configure the body measurements they desire to keep record of. All this can be found in Figure 80.

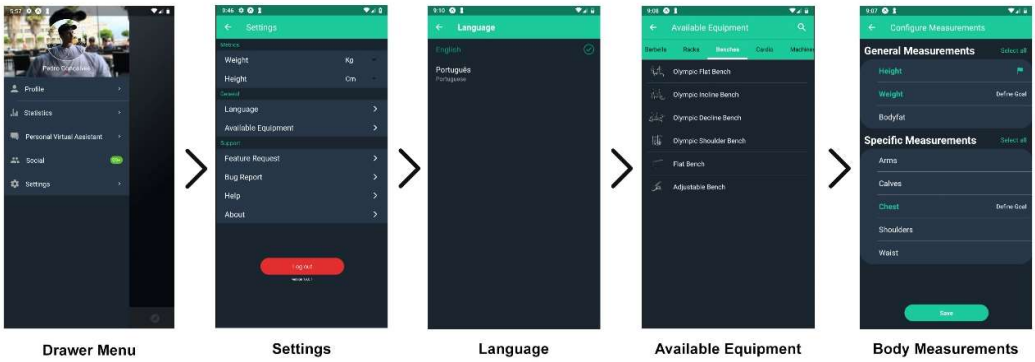

Figure 80 – Mobile App UI: Configurations

Even though the main flows of action were presented, there are still many more interesting and important screens, and also noteworthy intricacies regarding the development and implementation of the mobile application, which will all be addressed in the next section.

6.3 Server Application

In order to maintain a certain level of abstraction between the client's (mobile application) and the database, to serve as a middle man between them and other services and components, and also to compute complex operations that aren't recommended to be conducted in a client platform, a server application had to be devised.

The implementation of the server application was conducted using *Visual Studio* as the main IDE, and *C#* as the programming language, identical to what was used in the Recommender System.

The following sub-sections will be dedicated to the presentation of the intricacies and important characteristics and implementation decisions that were taken during the development of the server.

6.3.1 Authentication & Authorization

Authentication and Authorization are very important features to have, and ones predicted in the Non-Functional Requirements. To achieve this, the *ASP.NET Identity* [165] was used, which is a *Microsoft* tool that allows for an improved management mechanism for users and passwords, offering built-in classes that allow for the creation of both users and roles, as in Figure 81.

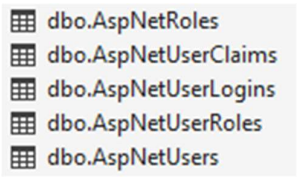

Figure 81 – *ASP.NET Identity* generated entities

It also offers templates "to add functionality to register, sing in and sign out a user." [165] Plus, to assure Authorization in the server – to only allow certain user's roles to access certain features (methods) – the *Authorize* attribute was used, as illustrated in Figure 82, which verifies in all *Controller* methods if the requests sent to the server are authorized to access the feature.

```
[Authorize]
[RoutePrefix("api/Programs")]
public class ProgramsController : ApiController
{
```

Figure 82 – *Authorize* attribute in the server's *Controller*

6.3.2 Encryption

Maintaining and securing sensitive data is a very important feature on a system that deals with such information. As such, an in order to answer some Non-Functional Requirements, an encryption tool was used to secure personal data such as the user's e-mail, body measurements and others.

Consequently, a class containing methods to encrypt and decrypt strings was devised. The methods were based on a built-in cryptography class named *RijndaelManaged.Rijndael* [165] which was selected by the *National Institute of Standards and Technology* as the candidate for the *Advanced Encryption Standard*, which is a "symmetric [method of encrypting text] chosen by the U.S. government to protect classified information and is implemented […] throughout the world to encrypt sensitive data." [166] It's safe to assume, then, that the algorithms used are "best practice".

Using the *RijndaelManaged* class to perform the encryption, allied with the *Rfc2898DeriveBytes* [165] function of the *Cryptography* built-in library (namespace *System.Security.Cryptography* [168]) which generates an encryption key using a key derivation function, *PBKDF2* [169], it's simple to encrypt a given string-based text with a string-based cypher key, which is an arbitrary string used to encrypt and decrypt, basically, a key.

6.3.3 Multi-language

Even though the multi-language was already assured and described in the mobile application, it is still required to address the database's multi-language. The database is supposed and prepared to hold huge amounts of information regarding training Programs, Plans, Workouts, Exercises and others, and having translations for all that data is a decision that has to be discussed thoroughly to make a good architectural decision so that performance is not compromised.

Usually, a certain domain class, i.e. Program, possesses their own attributes such as name, description, etc. But to support multiple languages, this approach cannot be taken. The class cannot, as well, be duplicated for every supported language because that would imply lots of redundant information.

As a solution, all the class's attributes that are desired to be translated were exported to a new class, containing those attributes and an attribute referring to the corresponding language. As such, there are multiple instances of this new class, one for each existent language, as illustrated by Figure 83, example (ii).

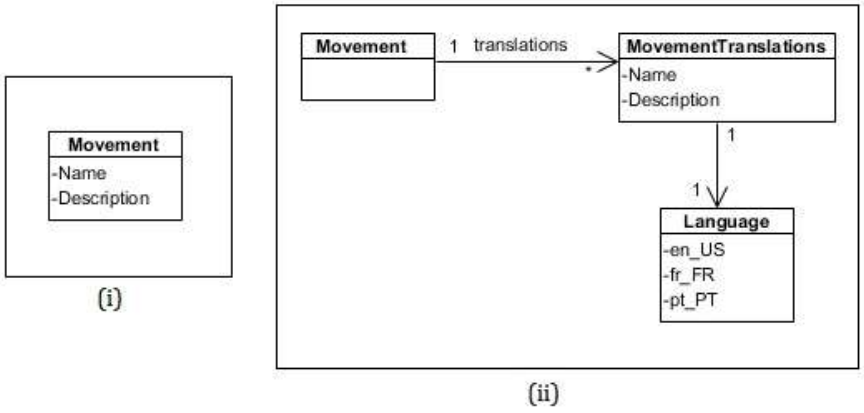

Figure 83 – Multi-language architecture solutions

In the previous figure, the first example (i) the *Movement* class contains two attributes that are desired to be translated in multiple languages – *Name* and *Description* – but with that implementation, the only way to achieve multi-language is to duplicate the class for each language. That is not a good solution, because if the class contains 10 more attributes that are not supposed to be translated, then that information will be duplicated in all instances of that class.

Since redundancy is not desired, a new solution was devised (ii). In this solution, the *Movement* class contains multiple instances of *MovementTranslations*, one for each supported language. Each instance contains the relevant attributes in a specific *Language*. That way, it's very easy to add new languages – add a new instance of *MovementTranslations* – and to retrieve information regarding a specific language - fetch the *MovementTranslation* instance where that *Language* is present.

This solution eliminates all redundancy, because only the attributes that are desired to be translated are "exported" from the main class, but the ones that are not stay there, only appearing once in the database.

6.3.4 Migrations

Migrations [171] "is the recommended way to evolve [an] application's database schema […]" [171] An evolving project usually implies changes in the database and model schema. With that, new classes, relationships, attributes, and other possible changes are always a possibility, and that means the database schema needs to be updated every time a small change occurs.

This is not very productive, and so, *Migrations* are used, as "a way to incrementally update the database schema to keep it in sync with the application's data model while preserving existing data in the database." [172] As such, Figure 84, contains the migrations used in the context of this project.

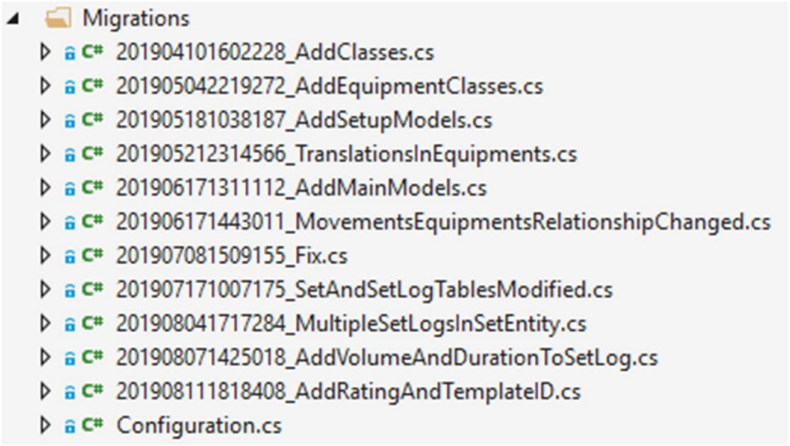

Figure 84 – Project's *Migrations*

6.3.5 Flow

Having presented all the server's characteristics and intricacies, it is now important to establish and present how it behaves when a request is received, as a way to better understand how it works as a whole. The flow of information, when a GET request is received, is illustrated by Figure 85.

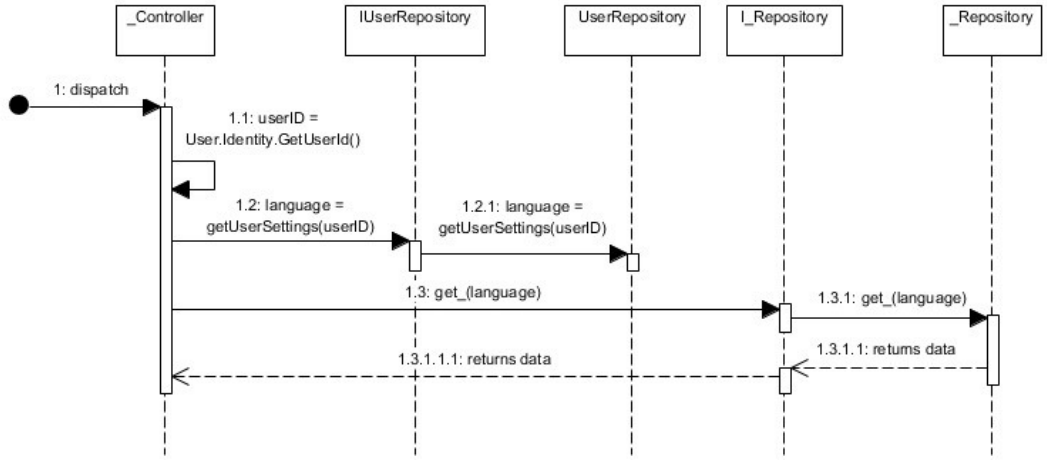

Figure 85 – Server's GET request flow

After the respective *Controller* receives the request, the currently logged user's ID is fetched, through *ASP.NET Identity*, which offers methods to easily retrieve these kinds of information. With the ID fetched, the user's language is obtained, by invoking the *getUserSettings* method, from *UserRepository*.

With all this information fetched, the server is now ready to process the specific information requested by the user, sending the user's language as argument to any *Repository* method, which then returns the desired information in the correct language.

6.4 Personal Virtual Assistant

One of the established ideas with value for the system was the implementation of a *Personal Virtual Assistant* (PVA), which was also designed as a functional requirement. This PVA besides offering an obvious innovative feature in the mobile fitness app market, it most importantly offers a way for users to communicate intelligently and have their issues and doubts answered from the comfort of their mobile devices.

For such a system to work, four components had to be involved which were the mobile application, where the user communicates and interacts with the PVA, the *ChatBot*'s API, to where all the message requests are redirected to formulate a response, the server application which returns database data, and also another API, from a *Microsoft* service called *LUIS* [173].

LUIS stands for *Language Understanding Intelligent Service* and is according to *Microsoft* "a cloud-based API service that applies custom machine-learning intelligence to a user's conversational, natural language text to predict overall meaning, and pull out relevant, detailed information." [174] Basically, it's a machine-learning service that translates natural language into apps, chat bots and IoT (Internet of Things) devices. This will help to build a *ChatBot*, the PVA, with intelligence to understand a respond to user requests.

An example on how all these components interact with each other can be viewed in Figure 86.

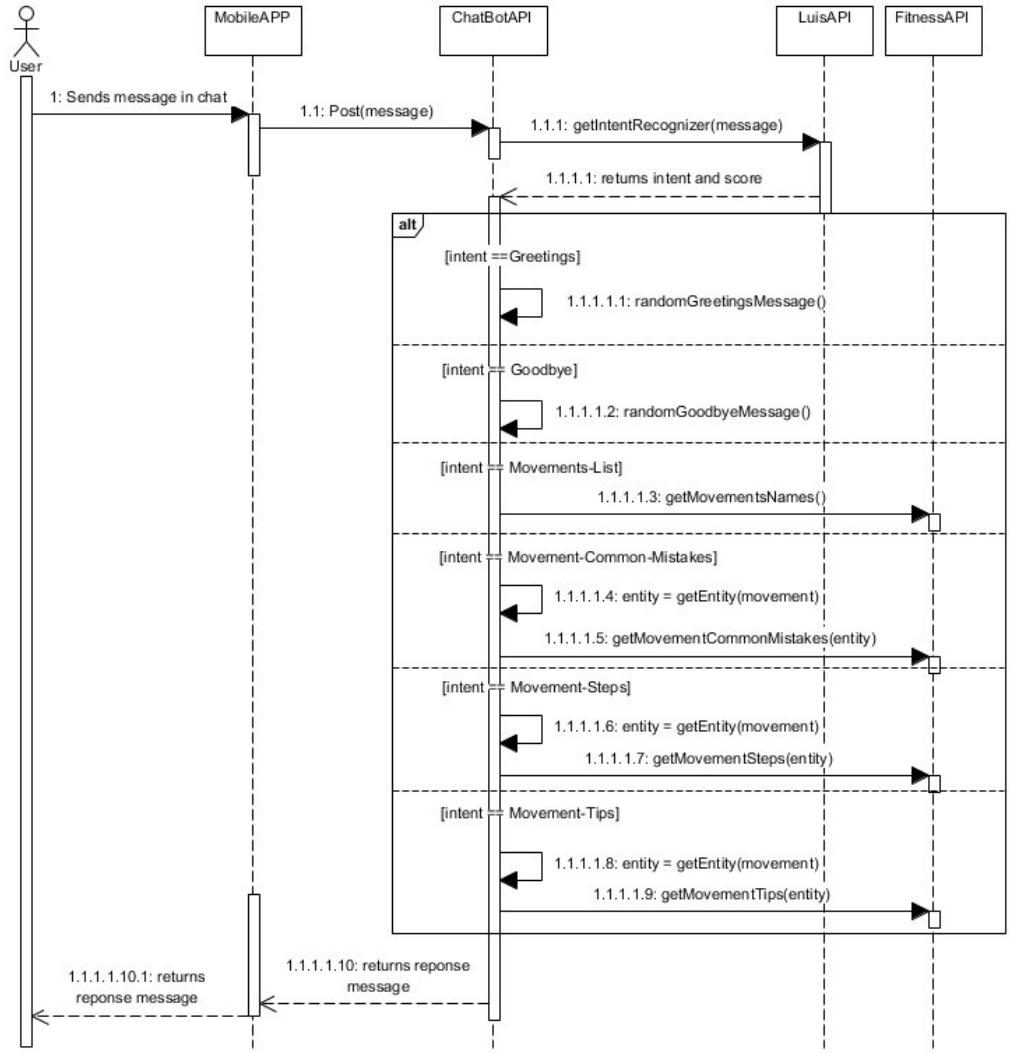

Figure 86 – PVA components communication

As the previous figure suggests, after receiving a message, the *LUIS*'s API first tries to recognize the intent of it. For example, a "hello" is intended to be a greeting, but for obvious reasons, a "give me a list of all exercises" has a different intent, and firstly it is important to recognize it. Deciphering a message's intent is not a very straightforward process because there can be multiple ways to greet someone – "hello" and "hi" have the same intent, which is to greet – and the service needs to understand and recognize intent as well as learn for future instances.

After understanding the intent of the message, the service can then decide which course of action to take. In the example of Figure 86, if the intent is "Greetings", the assistant will return a random greeting message, but if the intent is "Movements-List", the response should be the list of movements.

A bigger question now emerges which is "how to possess a model able to recognize custom intent?". The answer is that the model should be custom created with the desired intents to be recognized and then trained for it to be accurate and proficient in recognizing them. To do so,

LUIS offers a dedicated graphic interface (*LUIS portal*) to create the machine learning model. It is possible to start with a prebuilt model, build one from scratch or a combination of both by blending prebuilt pieces with custom information. [174] Regardless, after having the model, it is possible to easily created *intents* [175] and *entities* [176] which are a word or phrase to be extracted from the input message, or *utterance* [177] (view Figure 88), and then train the built model by manually reviewing *utterances* that *LUIS* had trouble deciphering the intent of and assigning them to the correct intents, which furtherly improves *LUIS*'s prediction capabilities.

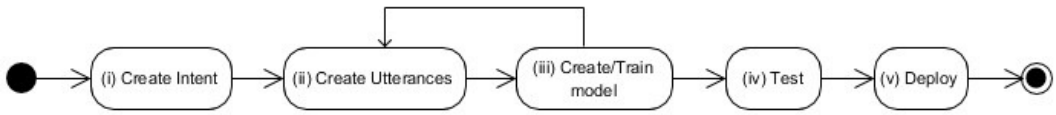

Figure 87 – Stages to build a *LUIS* model

In the previous Figure 87, the abovementioned stages for building a capable model are presented. Firstly, the *intents* where created, and for the context of this project, only six were chosen: *greeting, farewell, exercise list* and *exercise's common errors, tips* and *instructions*.

Afterwards, the examples of *utterances* need to be provided, so that the model has a baseline for recognizing intent from messages, as illustrated by Figure 88. Each *intent* needs multiple *utterance* examples. [174]

Example user utterance	Intent	Entities
"Book a flight to **Seattle**?"	BookFlight	Seattle
"When does your store **open**?"	StoreHoursAndLocation	open
"Schedule a meeting at **1pm** with **Bob** in Distribution"	ScheduleMeeting	1pm, Bob

Figure 88 – Example of extracting *intent* and *entities* from *utterances*[49]

As an example, the *utterances* used to train the model into recognizing the *intent* of "Movement-Tips" which is to return a specific movement's tips can be found in Figure 89, where the first part of the *utterance* is used to interpret *intent* and the last part – *movementName* – is the extracted entity. As also illustrated, "quero as dicas" has no recognized entity, and this will return a response from the PVA asking for the user to enter the desired entity.

[49] Adapted from xhttps://docs.microsoft.com/bs-latn-ba/azure/cognitive-services/luis/what-is-luis

Figure 89 – Movement-tips *utterances* examples

After having their *intents* and *entities* defined, and some *utterances* examples for the model to have a base, it is now ready to be trained. This process is an automatic process conducted by *LUIS* which will try to create a model to fit the entered information previously described.

After the training phase ends, the model is then ready to be tested. For that, some *utterances* are fed to the model in order to test if it can accurately recognize the intents. If, at some point, for a given *utterance*, the model cannot decipher the intent correctly, the phase for inputting *utterances* as baseline for each *intent* can be repeated, adding as many new *utterances* as desirable. Then the model is trained again and tested. These phases can be cycled as many times as desired until the model is satisfactory.

In the last stage, the trained and tested model is *deployed*, allowing now for the *ChatBot*'s API to communicate with *LUIS's API* to interpret user messages.

When a message is sent from the PVA to the *ChatBot*'s API, the first step is to invoke *LUIS*'s API to recognize the user's intent with the greatest *score*. The response of this request is a list of all existent *intents*, ordered by their *score* which is the likeliness of that being the user's *intent*.

Depending on the *intent*, the system can take different courses of action, as it was described previously. To organize that, an *IIntentHandler* interface was created, containing a method *HandleIntent* which all the instances of the interface should implement. Each *intent* has their own class, and therefore, each one implements the *HandleIntent* method. The behavior for each *intent* can be summed as follows:

- ***Greeting and Farewell***: Randomly selects a greeting or farewell message from existing ones.

- ***Exercise List***: Requests from the *FitnessAPI* the list of exercises.

- ***Exercise's Tips, Common Errors and Instructions***: After extracting the correct *entity* from the *utterance*, it invokes the *FitnessAPI* to return the tips, common errors or instructions for the given exercise.

After receiving the response, it is returned to the mobile application, where a chat built with the *gifted-chat* library was implemented, using the *Direct Line* [178] channel as a way to facilitate communication of the mobile application with the *ChatBot*'s API. This communication is what enables the user's *utterance* to reach the created model from *LUIS*'s API and receive a response afterwards, from the *ChatBot*'s API. This flow of information is illustrated in Figure 86.

```
onSend = messages => {
  this.setState({ messages: [...messages, ...this.state.messages] });
  messages.forEach(message => {
    directLine
      .postActivity(giftedMessageToBotMessage(message))
      .subscribe(() => console.log("success"), () => console.log("failed"));
  });
};
```

Figure 90 – Mobile application *Direct Line* usage

In the previous Figure 90, the communication with the *Direct Line* channel, from the mobile application, can be seen. Furthermore, an example interaction of the user with the PVA is illustrated by Figure 91.

Figure 91 – PVA interaction

6.5 Use Cases

In this section, the uses cases will be presented, in a way that allows for a better understanding on how the specified and designed requirements were implemented.

This is achieved through first describing the actions that must be performed by the user in order to utilize the respective functionality, with prints of the mobile app illustrating the process. Then, on a more detailed level, the implementation of the use case is described, with snippets of code from both the mobile app and the server, as well as a comprehensive written explanation of how everything ties together.

As a small side note, it was defined that the use cases involving "managing" can be divided into three action categories: *create*, *edit*, *delete*. Due to time restrictions, not all actions can be implemented for all use cases, which means that some should be prioritized. For proper functioning, the only fundamental one is the *create* action, and as such, it will be the one prioritized. The others, even though they were designed, won't be addressed in this section for the mentioned reasons.

6.5.1 US02: View training Plan/Program templates

From the *Discover* screen, depicted in Figure 92, the user can decide to view the template list of Plans and Programs. When selecting the "*See all"* option, all the Plans/Programs are shown, in their respective screen. At the same time, the user can also select a specific program directly from the *Discover* screen's interface.

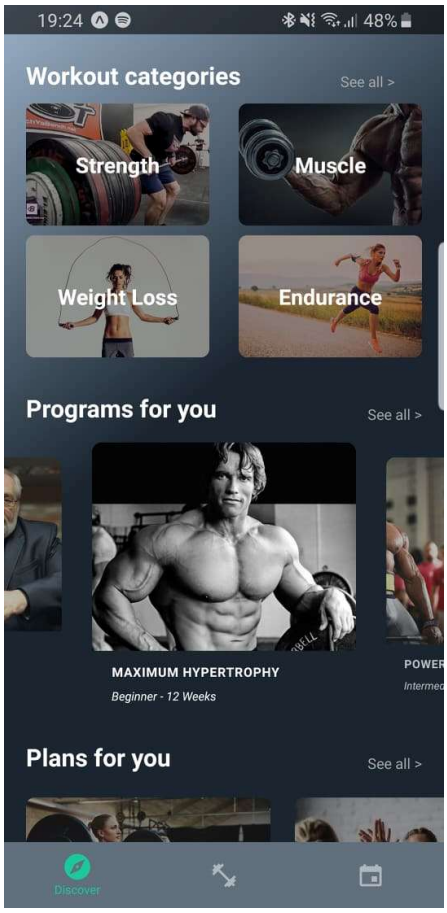

Figure 92 – Discover screen

When the "S*ee all*" option is selected, the user is redirected to the respective Plan or Program list screen. There, before rendering the screen, it is verified if the list is already loaded in the system, because if it is not, the information needs to be fetched from the server.

```
if (isEmptyList(this.props.programs)) {
    if (isEmptyList(this.props.equipments) || isEmptyList(this.props.muscleGroups)) {
        this.props.onGetProgramsAndFilters().then((result) => {
            this.apiRequestBodyHandler(result);
        });
    } else {
        this.props.onGetPrograms().then((result) => {
            this.apiRequestBodyHandler(result);
        });
    }
} else {
    this.setState({
        programs: JSON.parse(JSON.stringify(this.props.programs))
    });
}
```

Figure 93 – Program list validation

The said validation is depicted in Figure 93, where it is firstly verified if the *Redux*'s variable for the list of Programs is empty. If it is not, then, the screen's state variable is updated with that information, which means that no connection with the server is required. On the other hand, if the list is empty, before requesting the server, it is verified if *Redux* also has information regarding the equipment and muscle groups. This information is required in the context of the Programs screen for filtering purposes – one has to have information regarding muscle groups to be able to filter Programs with it. Depending on the result of this verification, one of two methods is called – *onGetProgramsAndFilters* and *onGetPrograms*. The difference between them is that the latter only retrieves the list of Programs, and the first also returns the list of filters (equipment and muscle groups).

```
export const getPrograms = () => {
    console.log("## Get Programs action ##");
    return async (dispatch, getState) => {
        let returnResult;
        const token = getState().auth.token;

        dispatch(uiStartLoading());

        await fetch(FITNESS_API + "api/Programs", {
            method: "GET",
            headers: {
                "Authorization": token,
                "Content-Type": "application/json"
            }
        })
            .then(res => {
                if (!res.ok) {
                    console.log(res);
                    throw new Error("Server error.");
                }
                return res;
            })
            .then((res) => (res.json()))
            .then(parsedRes => {
                if (parsedRes || parsedRes === []) {
                    dispatch(setPrograms(parsedRes));

                    returnResult = { data: parsedRes, success: true };
                } else {
                    throw new Error("Error getting info from server.");
                }
            })
            .catch(err => {
                console.log("Error: " + err.message);
                returnResult = { data: err.message, success: false };
            });

        dispatch(uiStopLoading());
        return Promise.resolve(returnResult);
    };
};
```

Figure 94 – *getPrograms* action

Each of these methods dispatch a different action, and the one that fetches only the list of programs from the server can be found in Figure 94, where the *API*'s *GET* method for the *Programs Controller* is called, which returns the desired Program list. This method can be found in Figure 95, where the *GetPrograms* method in the *Program Repository* class is invoked, passing as arguments the ID of the current user, and their preferred language, in order to return the list in the correct language.

```
public async Task<List<ProgramDTO>> GetPrograms()
{
    string userID = User.Identity.GetUserId();
    UserSettingsDTO userSettings = await userRepository.GetUserSettings(userID);

    return await programRepository.GetPrograms(userID, userSettings.Language);
}
```

Figure 95 – *GetPrograms* server method

After receiving the result, the *Redux* action sets the state of the store by dispatching another action – *setPrograms* – which can then be accessed by the screen. While waiting, the user is presented with a loading screen. The resultant screen can be found in the first two images from Figure 78.

6.5.2 US03: Manage training Plan/Program

Managing a training Plan or Program can imply one of three actions – create, delete, or edit. As described previously, the *create* action was the priority, and due to time restrictions, it was the only one implemented.

To create a Plan or a Program, the process is relatively similar and by explaining the creation of a Program, the Plan's will also be addressed indirectly, which is the reason for combining the two in a single use case.

In the Domain Model section, it was described how a Program is characterized by a name, description, list of plans, and other variables such as its goals, level, etc. The Plan, on its turn, is characterized by a list of workouts, a name, a description, and other variables too. Each Workout has its name and the exercise list that composes it. And each exercise has the number of sets that define it, each one with the reps, weight, intensity and rest time, depending on the type of exercise. This can be seen in the next Figure 96.

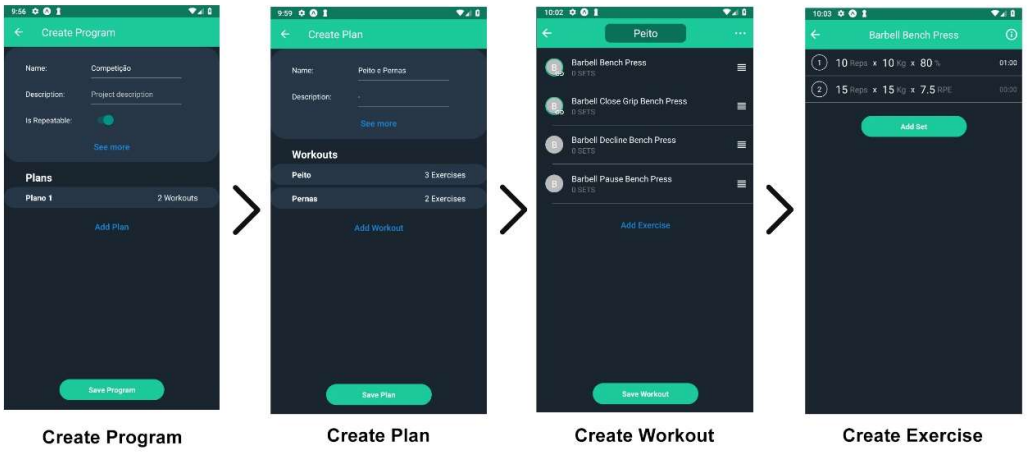

Figure 96 – Create Program screen flow

When in the *Create Program* screen, the user is prompted to enter the required information and to configure the list of Plans that make up the desired Program. To configure each Plan, the user is redirected to the *Create Plan* screen, which is also the same one used for creating single Plans, without any Program associated. The same happens when configuring each Workout, the user is redirected to the *Create Workout* screen, which is also the same one used for creating *Single Workouts*. There, the user selects the desired exercises, and for each one, it can configure the number of sets, the reps, the intensity, rest time, etc.

To enter the Program's *goals* and *training types*, and the Plan's *training types* the user is presented with an interface that allows them to enter the desired ones, as in Figure 97.

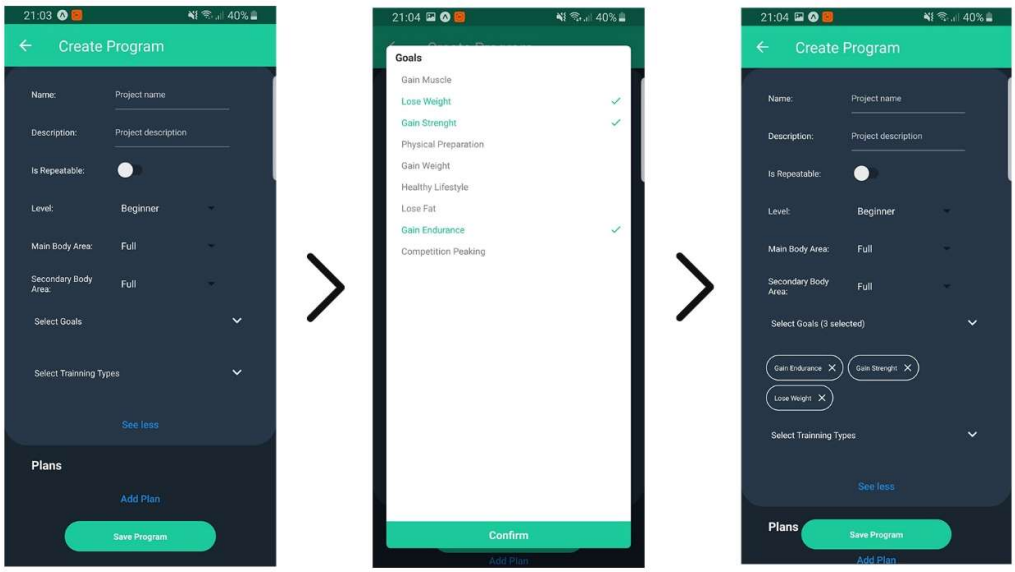

Figure 97 – Configure Program Details

After all the information is entered, from the Program's name to each of the Exercise's sets, a JSON object, containing all this information, is built and sent as an argument to a function that dispatches the *createProgram* action. This action calls the *Programs Controller* through a POST request, sending the JSON object as argument.

```
[ResponseType(typeof(ProgramDTO))]
public async Task<List<ProgramDTO>> PostProgram(ProgramDTO program)
{
    if (!ModelState.IsValid)
    {
        return new List<ProgramDTO>();
    }

    string userID = User.Identity.GetUserId();
    UserSettingsDTO userSettings = await userRepository.GetUserSettings(userID);

    foreach (PlanDTO plan in program.Plans)
    {

        for (int i = 0; i < plan.Workouts.Count; i++)
        {
            WorkoutDTO workout = plan.Workouts.ElementAt(i);

            IEnumerable<int?> supersets = GetExercisesSupersets(workout.Exercises);

            int supersetsLength = supersets.Count();
            if (supersetsLength > 0)
            {
                List<SuperSet> createdSupersets = await supersetRepository.CreateSuperSets(supersetsLength);
                workout = SetSupersetsInExercises(workout, supersets, createdSupersets);
            }
        }
    }

    //goals & trainning types
    List<Goal> goalsList = await goalRepository.GetGoalsFromID(program.Goals.Select(x => (GoalType)x.ID).ToList());
    List<Training> trainingList = await trainingTypeRepository.GetTrainningTypesFromID(program.TrainingTypes.Select(x => (TrainingType)x.ID).ToList());

    var programModel = program.toCreateModel(userID, goalsList, trainingList);
    await programRepository.CreateProgramAsync(programModel);

    return await programRepository.GetPrograms(userID, userSettings.Language);
}
```

Figure 98 – *PostProgram* method

The previously mentioned function is depicted in Figure 98. Firstly, the user settings are fetched from the *User Repository*, which will be helpful when returning the list of programs in the

correct language. The *foreach* block of code is very important because it deals with the existence of *Supersets*. These are very special and require a particular type of attention. Each Plan and each Workout is iterated, and for everyone, the *GetExercisesSuperSets* method is invoked, which it iterates their exercises in order to obtain a list of *integers* correspondent to the IDs of the *Supersets* created in the app. Then, if the method in Figure 99 is executed, which replaces the attribute *SuperSetId* of every exercise that is grouped to a *superset*, with the one returned from the database.

```
private static WorkoutDTO SetSupersetsInExercises
    (WorkoutDTO singleWorkout, IEnumerable<int?> supersets, List<SuperSet> createdSupersets)
{
    int length = supersets.Count();
    for (int i = 0; i < length; i++)
    {
        int ssID = (int)supersets.ElementAt(i);
        int createdSsID = createdSupersets.ElementAt(i).ID;

        singleWorkout.Exercises.Where(x => x.SuperSetId == ssID).ToList().ForEach(s => s.SuperSetId = createdSsID);
    }

    return singleWorkout;
}
```

Figure 99 – *SetSupersetsInExercises* method

Then, the model of the Program needs to be created to be sent to the *CreateProgramAsync* function from the P*rogram Repository* which will add it to the database. This model is created using a type adapter.

```
public static Program toCreateModel(
    this ProgramDTO program,
    string userID,
    List<Goal> goals,
    List<Training> trainningTypes)
```

Figure 100 – Program model creator method

The *PostProgram* function, in the end, returns the complete list of Programs, to be received by the mobile app, which will update its *Redux*'s store variable relative to the program list, so that the information is consistent. This update is conducted though the dispatchment of an action called *setPrograms*, already described previously.

As a final note, it is important to note that, by the way things are implemented and defined, a Plan, in terms of database modeling, is always part of a Program. That way, there's only one method to save Plans and Programs. The distinction comes from the fact that a Program with only one Plan is what constitutes, in terms of domain logic, a Plan.

6.5.3 US04: Manage Body Measurements

In the user's profile, the user can access the *Body Measurements* screen, which when it's being opened by the first time, will present not the main screen but the *Configure Measurements* one. As it is the user's first time accessing their body measurements, they have yet to be configured. This configuration refers to the selection of body measurements that the user desires to keep track of. If they don't want to log information regarding, for instance, their body fat, then there's

no point in showing it at all. As such, the user selects the desired body measurements, from the list presented, as shown in Figure 101, where the *Height*, *Calves*, *Chest*, and *Shoulders* are already selected. In this screen, the user can also define a goal to associate with the respective body measurement, illustrated by the "green flag" present in the *Height* and *Chest* measurements.

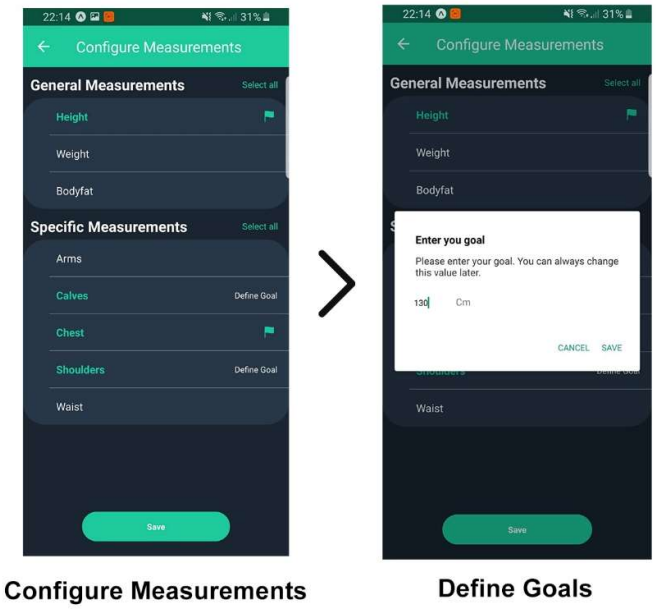

Figure 101 – Configure Body Measurements

After selecting the desired measurements and saving, the method in Figure 102 is invoked. Here, the function *onPutSelectBodyMeasurements* is called, receiving as argument the object containing the selected body measurements objects. This function, on its turn, dispatches a *Redux* action named *putSelectedBodyMeasurements* which also receives the list of body measurements as argument. In this action, a PUT request is sent to the *BodyMeasurements Controller* in the server.

```
persistMeasurementsInfo() {
    this.props.onPutSelectedBodyMeasurements(this.state.selectedMeasurements).then(() => {
        this.navigateToScreenDestroyStack('BodyMeasurements');
    });
}
```

Figure 102 – Save selected Body Measurements method

The method which receives the request is partially shown in Figure 103. The full code is not present in the figure because it was too extensive. As such, it only represents the main flow behind the actual method.

Regardless, firstly the current user is fetched from the respective repository, which then allows to retrieve their body measurements information. The received by argument list of body measurements is then iterated in a *foreach* loop and for each one it is verified if there is already a measurement in the user's list with the same type. There can only be one type of

measurements per user – if the user's list already contains a *Chest* type measurement, it cannot contain another. And so, if there is no object of the current measurement's type then it is created using the *CreateBodyMeasurement* function from the *Body Measurements Repository*, which simply adds it to the database.

If, on the other hand, the current body measurement already exists in the user's list, then it is verified if they possess the same ID. This is to confirm that they represent the same object, because if they don't have the same ID, it means that in the mobile app, the user removed the body measurement and then added it again, which resets its ID and loses all its logs as well.

Consequently, if they don't possess the same ID, the one sent from the mobile app is added to the database and the one that was in the user's list is removed also from the database, maintaining the consistency of only existing a maximum of one measurement type per user. The used method to do so is named *CreateAndDeleteBodyMeasurements* and receives both measurements as argument.

At the same time, if they do in fact possess the same ID, it means they are the same object, and there's no need to add nor delete a measurement to or from the database, respectively. All there remains is to handle the measurement's logs. If it's the first "save" of body measurements configurations, there are no logs yet, and so, the method ends here. This happens because the method is used for configuring body measurements as well for adding logs to them, as it will be described next.

```
[ResponseType(typeof(void))]
public async Task<List<BodyMeasurementsDTO>> PutBodyMeasurements(List<BodyMeasurementsDTO> bodyMeasurements)
{
    ApplicationUser user = await userRepository.GetUserByIdAsync(User.Identity.GetUserId());

    ICollection<BodyMeasurements> initialBodyMeasurements = user.Profile.BodyMeasurements;

    foreach (BodyMeasurementsDTO bodyMeasurement in bodyMeasurements)
    {
        BodyMeasurements fetchedBodyMeasurement = initialBodyMeasurements
            .FirstOrDefault(x => x.BodyMeasurementType.Equals(bodyMeasurement.BodyMeasurementType));

        if (fetchedBodyMeasurement != null)
        {
            if (fetchedBodyMeasurement.ID == bodyMeasurement.ID)
            {
                List<BodyMeasurementLog> newLogs = new List<BodyMeasurementLog>();
                foreach (BodyMeasurementsLogDTO log in bodyMeasurement.Logs)
                {
                    if (!fetchedBodyMeasurement.BodyMeasurementLogs.Any(x => x.ID.Equals(log.ID)))
                    {
                        newLogs.Add(new BodyMeasurementLog
                        {
                            Date = System.DateTime.Now,
                            Value = new BodyMeasurementValue { Value1 = log.Value.Value1, Value2 = log.Value.Value2, GeneralMetricType = metricType },
                            BodyMeasurementsId = fetchedBodyMeasurement.ID
                        });
                    }
                }

                fetchedBodyMeasurement.Goal.Value = bodyMeasurement.GoalValue;
                await bodyMeasurementsRepository.UpdateBodyMeasurement(fetchedBodyMeasurement, newLogs);
            }
            else
            {
                await bodyMeasurementsRepository
                    .CreateAndDeleteBodyMeasurement(bodyMeasurement.createToModel(user.Profile.ApplicationUserId, metricType), fetchedBodyMeasurement);
            }
        }
        else
        {
            await bodyMeasurementsRepository.CreateBodyMeasurement(bodyMeasurement.createToModel(user.Profile.ApplicationUserId, metricType));
        }

    }

    List<BodyMeasurements> result = initialBodyMeasurements.Where(p => !bodyMeasurements.Any(p2 => p2.BodyMeasurementType == p.BodyMeasurementType)).ToList();
    await bodyMeasurementsRepository.RemoveBodyMeasurementsAsync(result);

    return await bodyMeasurementsRepository.GetBodyMeasurementsList(user.Id, user.Setup.Language);
}
```

Figure 103 – *PutBodyMeasurements* method

After the information is saved, the user is redirected to the main *Body Measurements* screen, which will contain the list of the selected body measurements, with the defined goals and with logs yet to be entered. The user can then select the option to add logs, which will redirect them to the first screen in Figure 104. This screen contains solely the list of the user's selected body measurements, and for each one, the option to enter a value to be logged. Some measurements, as the *Calves* one, are special because they can have not one, but two entry logs. In real world context, they refer to bilateral body parts, for example the arms, which can have a measure for the right and left side. After the user enters the desired information and saves it, the body measurements list, with its respective logs, is sent as argument to the already described putSelectedBodyMeasurements action.

As it was previously mentioned, the method of Figure 103 is used to configure and to add new body measurements entries. And so, after saving the logs, the same method is invoked, and when it reaches the part where it stopped before - after validating that the currently being iterated body measurement and the one fetched from the user's list are the same - the logs are iterated, and for each one it is verified if they already exist. If they do, nothing happens, but if they don't, a new *BodyMeasurementLog* object is created with the correct information and added to the respective log list, which will then be updated in the database through the *UpdateBodyMeasurement* method.

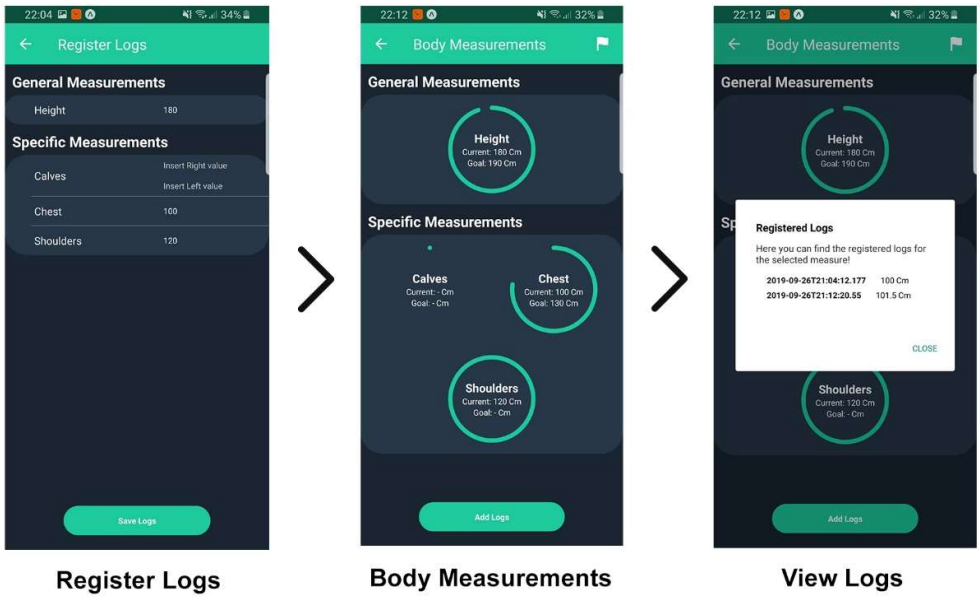

Figure 104 – Manage Body Measurement Logs

After adding logs, the user is redirected to the main screen again, where it would show something like the second screen in Figure 104, where a progress circle indicates how far from the goal the last entry log of that specific body measurement is. If any body measurement is selected, a dialog appears, showing the list of logs, with the respective values and associated date. The date is added when creating the *BodyMeasurementLog* object in the server with the current date and time (see Figure 103).

Finally, the user, if desired, can re-configure the list of body measurements by selecting the icon on the right of the screen's header ("white flag" icon), which will redirect them to the configuration screen, with the existent configurations being shown.

6.5.4 US08: View training Logs

In the bottom tab, the user can quickly access their log history. This screen, present in Figure 105, contains the Agenda of the user, with an interactive horizontal top calendar, centered in the current day and showing an entire week, and a bottom scrollable calendar, with the details for each day presented. The top calendar can be expanded to visualize entire months' worth of days, as illustrated by the second image in Figure 105. The blue dots in each day represent a Workout conducted on that specific day. For example, in the 13th day, the *Volume Day* workout was conducted, which is part of the *Hypertrophy Block* Plan and the *Off-Season Progressive Overload* Program. It's also noticeable the volume of that workout, *1854kg*, and the duration, *1h35m20s*.

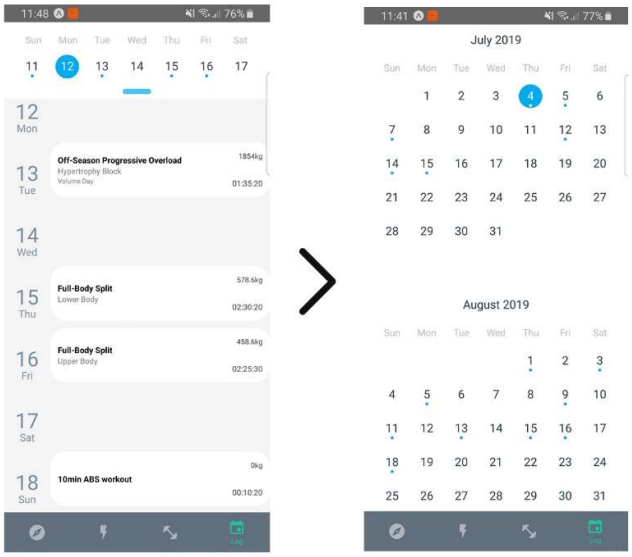

Figure 105 – Log screen

The information regarding workout logs is only imported to the app if the screen is requested to be opened. This is to avoid overcharging the app with too much unnecessary information when it is first starts. That way, the user doesn't have to wait long before being able to use the app.

Regardless, when the screen indeed is opened, a *Redux* action is dispatched, fetching from the server the list of workout logs, using the method of Figure 106, which is in the *Logs Controller*. There, the user's logs are fetched from the *Logs Repository*, through the *GetLogsByDay* method, which returns them by their date.

```
[Route("getLogsByDay")]
public async Task<List<LogDTO>> GetUserLogsByDate()
{
    string userID = User.Identity.GetUserId();
    UserSettingsDTO userSettings = await userRepository.GetUserSettings(userID);

    return await setLogRepository.GetLogsByDay(userID, userSettings.Language);
}
```

Figure 106 – *GetUserLogsByDate* method

The list is then received by the mobile app, which updates its *Redux* store variable by dispatching a *setLogs* action. In the screen, this information is accessed and the objects to be fed to the calendar library are built, as illustrated by Figure 107, and finally, the calendar is shown to the user.

```
buildItems(data) {
  var newItems = {};

  data.forEach(element => {
    newItems[element.Date] = [];
    newItems[element.Date].push({
      programName: element.ProgramName,
      planName: element.PlanName,
      workoutName: element.WorkoutName,
      singleworkoutName: element.SingleWorkoutName,
      duration: element.Duration,
      volume: element.Volume
    });
  });

  this.setState({ logs: newItems });

}
```

Figure 107 – Log objects' builder

6.5.5 US10: View Statistics

It was desired to provide the user with statistics related to training progress, body measurements, and others, but due to time restrictions it was only implemented statistics referring to body measurements, more specifically bodyweight. These statistics would be all very similar in terms of structure, so by fully implementing one of them the system is then ready to support other types' as well.

When accessing their Profile, the user can promptly view a graph representing the evolution/progress of their bodyweight, as illustrated by Figure 108. If the "*white arrow*" on the right side of the graph is pressed, the user is redirected to their body measurements screen, as described in the US04: Manage Body Measurements section.

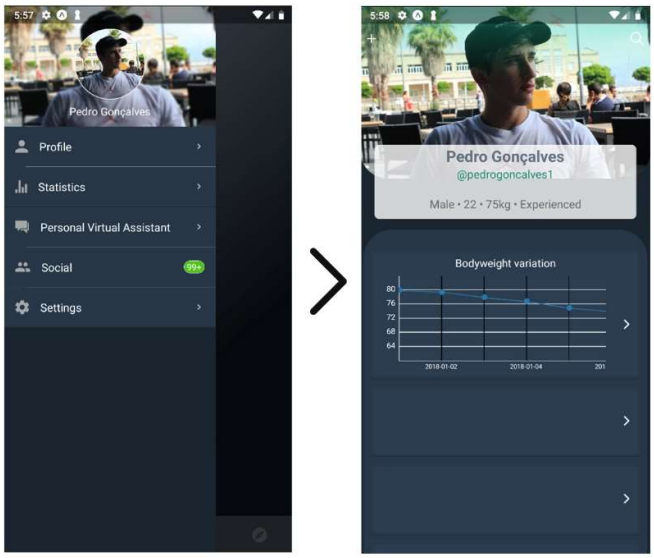

Figure 108 – Profile screen statistics

To construct such graph, when the user Profile is loaded, their body measurements should be loaded too, if they weren't already. In Figure 109, the method responsible for loading that information is observable. This method first verifies if the body measurements list on the *Redux*'s store is empty, and if it is, the *onGetSelectedBodyMeasurements* method is invoked, which dispatches the *getSelectedBodyMeasurements* action, retrieving the list of body measurements. If this list is empty, it means that the user has yet to configure their body measurements, which will be important for when they try to open the Body Measurements screen.

```
componentDidMount() {
  i18n.locale = this.props.language;

  if (isEmptyList(this.props.selectedBodyMeasurements)) {

    this.props.onGetSelectedBodyMeasurements().then((result) => {

      if (isEmptyList(result.data) && result.success) {

        this.setState({ isFirstTime: true });
      }
    });
  }
}
```

Figure 109 – *componentDidMount* method from Profile screen

The graph itself was built using the information regarding the logs of bodyweight (it can be of other types), which have a value, the Y-axis, and a date, X-axis.

As a final note, it is important to notice that the used library to build the graph (*react-native-pure-chart)* wasn't ready to be used from the start and had to be deeply modified. A GitHub

issue post[50] was created, explaining those changes. These include the addition of customizing the Y-axis values to start and end near the bottom and top limits of the logs' values. All the code and written explanation can be found on the opened issue.

6.5.6 US15: Change Language

Multi-language is supported by the system, and as such, a user can decide to change their preferred language. To do so, the *Language Screen* can be accessed through the *Settings*. After that, the user is presented with the list of available languages to change to, and after selecting one, it is asked to confirm the decision before saving the information, as illustrated by Figure 110. This is to prevent changing the entire app's language by accident.

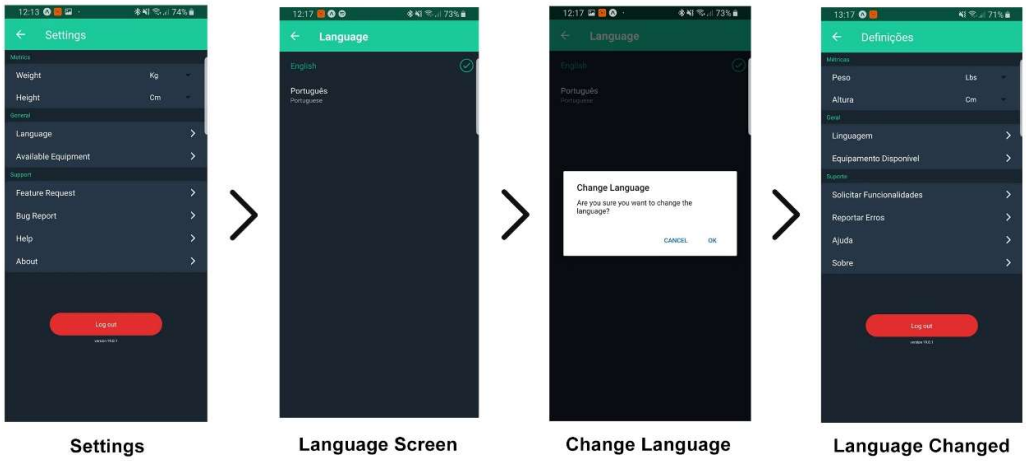

Figure 110 – Change system's language

Additionally, it is worth noting that the user's language not only affects the mobile app's language, but it also of all the retrieved information from the database – for instance, when retrieving the list of programs, or the list of training types, if the user's language is Portuguese, then the information will be retrieved in Portuguese, as it was described in the Server Application section.

Furthermore, after confirming to change the system's language, the *onChangedLanguage* function is invoked, which on its turn dispatches the *updateLanguage Redux action*, as depicted in Figure 111.

```
const mapDispatchToProps = dispatch => {
    return {
        onChangedLanguage: locale => dispatch(updateLanguage(locale))
    };
};
```

Figure 111 – change language function to redux action

[50] https://github.com/oksktank/react-native-pure-chart/issues/82

The *action*, then, calls the server through *SetupsController* and invokes the method shown in Figure 112, where the new language is received as argument. The method, firstly, fetches the current user's object, edits it with the new language and saves the changes through the *EditUser* method from the *User Repository*.

```
[HttpPut]
[Route("updateLanguage/{language}")]
public async Task<IHttpActionResult> PutUserLanguage(string language)
{
    ApplicationUser user = await userRepository.GetUserByIdAsync(User.Identity.GetUserId());
    LanguageType languageCode = EnumEx.GetValueFromDescription<LanguageType>(language);
    if (languageCode == LanguageType.None)
        return NotFound();

    user.Setup.Language = languageCode;
    IdentityResult status = await userRepository.EditUser(user);
    if (!status.Succeeded)
        return BadRequest();

    return StatusCode(HttpStatusCode.NoContent);
}
```

Figure 112 – *PutUserLanguage* method

After the *action* receiving the return answer from the server, it dispatches two new actions – *setLanguage*, which sets the *Redux*'s store language variable to the new one, in order for other screens to be able to correctly access it, and *clearInformation*, which clears the *Redux*'s variables for the available equipment, programs, training types, and others. For instance, if the available equipment is in English and the user just changed the language to Portuguese, the list is inconsistent, and as such, it is cleared so that when it is reloaded it can be fetched with the correct language.

6.5.7 US16: Change Unit System

Depending on the user's preference and their localization, they might favor a certain unit system over others. For example, someone in the USA, Burma or Liberia, the imperial system is the adopted one, which according to the *Central Intelligence Agency*'s *"The World FactBook"* [165], are the only countries in the world that still aren't using the metric system.

Furthermore, if the user prefers the metric system, but the gym he trains only has pound plates, they might want to change their preferred weight unit of measure to lbs. instead of kgs.

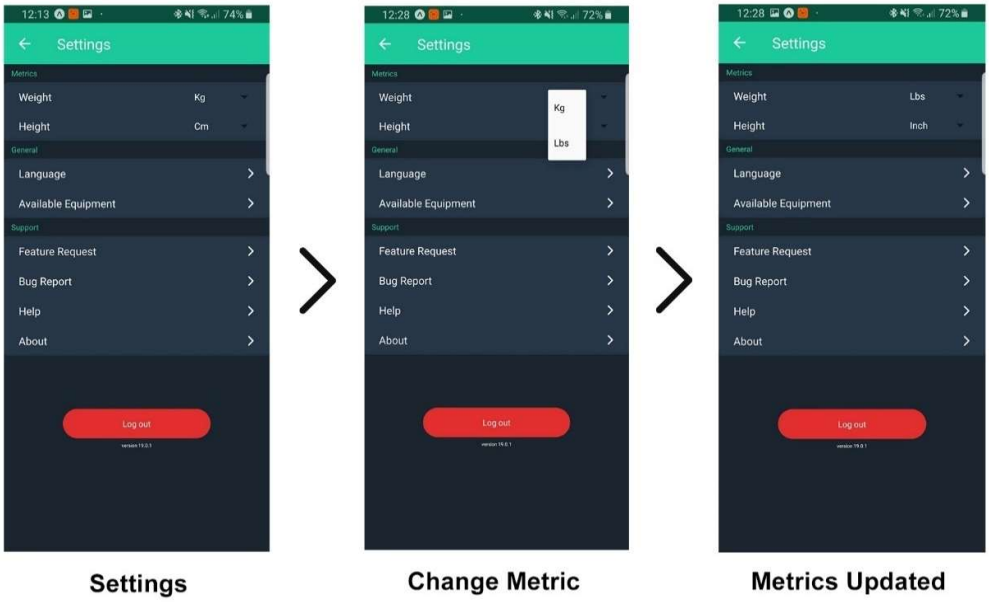

Figure 113 – Change unit system

In the previous Figure 113, it is observable the full action of changing between unit systems. By default, the metric system is adopted, but just bey pressing the desired unit – weight or height – the user can configure their preferences. Every time a user changes a unit, the method in Figure 114 is invoked, calling a *onSetUnit* function that dispatches the *Redux*'s action responsible for updating the user's units in the server, calling it through the *Setups Controller*'s *updateUnits* method, which receives the unit type and its new value as arguments and updates the user's object in the database with the new information.

```
_onUnitListChangedHandler = (unitType, unitValue) => {
  this.props.onSetUnit(unitType, unitValue).then((result) => {

    if (!result.success) {
      alert(result.data);
    }
  });;
}
```

Figure 114 – Unit system change handler

After receiving confirmation of the success of the operation, the *Redux action* dispatches a *storeUnits* action which will update the *Redux*'s store variables and also the *AsyncStorage*, as illustrated by Figure 115, where the *setBothUnits* action is dispatched, updating the height and weight *Redux*'s variables, and invoking the *setAsyncProperty* for both the height and weight units, updating also the *AsyncStorage*.

```
/** Save user units info in Async Storage and Redux. */
export const storeUnits = (height, weight) => {
    return async dispatch => {
        dispatch(setBothUnits(height, weight));

        await setAsyncProperty(USER_UNIT_HEIGHT, height);
        await setAsyncProperty(USER_UNIT_WEIGHT, weight);
    };
};
```

Figure 115 – *storeUnits* action

6.5.8 US17: Manage Available Equipment

Each user might have access to different equipment, and that should impact things like their exercise selection and also recommendations – if a user doesn't have access to a specific specialized barbell, there shouldn't be any exercises recommended to them that require that piece of equipment, and the user should be able to filter the exercise list for ones that they are able to perform.

To configure the available equipment list, the user can access the dedicated screen under the *Settings Screen*. This is illustrated by Figure 116, where after loading the available equipment list, the user is presented with a screen with the list divided by equipment types, such as *Barbells*, *Machines*, *Benches*, and other types, to facilitate searching. It is also possible to view the equipment details, such as an expanded view of its image, by clicking on it.

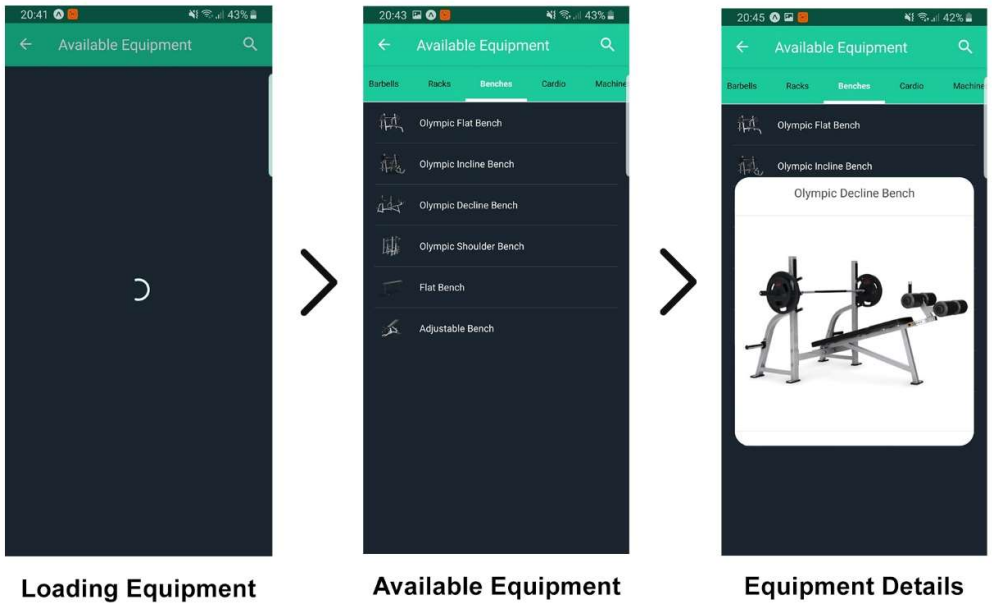

Figure 116 – Open Available Equipment screen

When the screen is opened, the snippet in Figure 117 is executed, which firstly verifies if the *Redux*'s store variable for the available equipment list already contains information. If it does, then the screen's state is updated, and if not, the *onGetEquipmentByCategory* method is

invoked, which dispatches *getEquipmentsByCategory* action. This action makes a server call through the *EquipmentCategory Controller*, which returns a list of *EquipmentCategory*. This object is characterized by an attribute identifying the category and the list of equipment of that category. That way, the information is returned organized by category, making it easier to separate in the aforementioned tabs. If the result is successful, the screen's state is also updated.

```
if (this.props.availableEquipments.length === 0) {
    this.props.onGetEquipmentsByCategory().then((result) => {

        if (!result.success) {
            alert(result.data);
        } else {
            this.setState({
                availableEquipmentList: JSON.parse(JSON.stringify(this.props.availableEquipments)),
            });
        }
    });
} else {
    this.setState({
        availableEquipmentList: JSON.parse(JSON.stringify(this.props.availableEquipments))
    });
}
```

Figure 117 – Get available equipment list snippet

After the information is loaded, the user can then decide to select some equipment to add to their personal list. As illustrated by Figure 118, the user can search for equipment in the search bar, which also shows the number of found results. After searching, they can select as many as desired and by saving, the list is automatically updated, both in the app and the server.

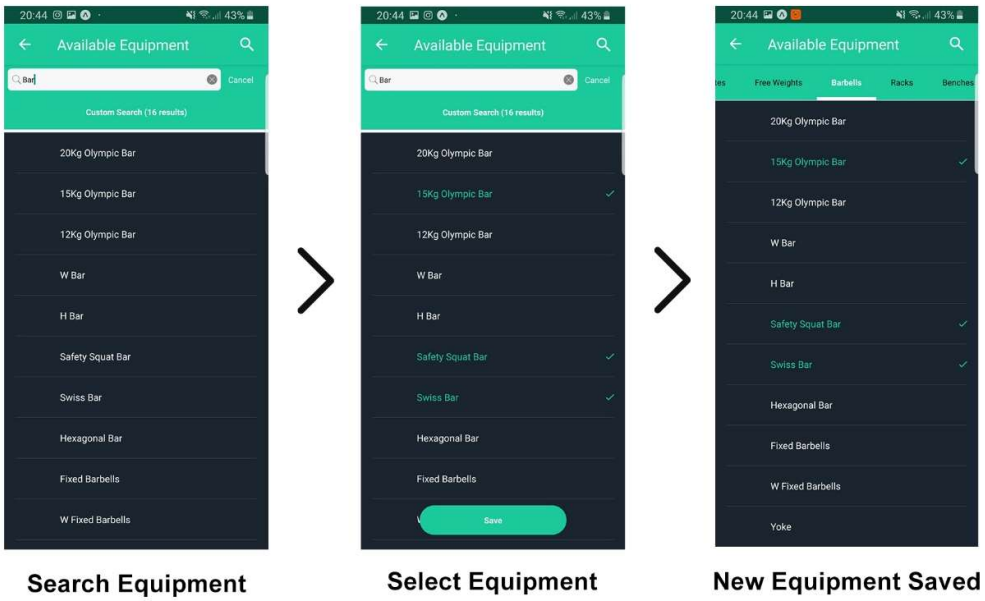

Figure 118 – Filtering and Adding equipment

To accomplish this, every time an equipment is selected, it is added to a temporary list that if the user decides to save it is sent as argument to the *onSetUserAvailableEquipment* method

(see Figure 119), which dispatches the *setUserAvailableEquipments* action. This action calls the server through the already mentioned *EquipmentCategory Controller*, which saves the received information in the database and returns if the action was successful or not. If it was, the *Redux*'s variable for the list of equipment is updated (still in the action), and finally, in the method of Figure 119, it clears the changed equipment list, given that all changes were already saved.

```
_saveAvailableEquipment(next) {
    this.props.onSetUserAvailableEquipment(this.state.changedEquipment, this.state.availableEquipmentList).then((result) => {

        if (!result.success) {
            alert(result.data);
        } else {
            this.setState({ changedEquipment: [] });
        }

        next();
    });
}
```

Figure 119 – Save available equipment method

Because the information is saved in the *Redux* store, if the user quits the screen and opens it again, the information won't be fetched from the server again, eliminating the need for the user to wait again and have instant access to the information.

6.6 Non-Functional Requirements

The entire system was devised and implemented in order to answer the designed requirements, both the functional and non-functional. Given that the first were already described, through the use cases, the latter need to be addressed as well, which is what this section is dedicated to – summing the information that was already presented in previous sections in a way that is easier to understand how that would answer the designed non-functional requirements.

Next, the most important non-functional requirements will be discussed in the perspective of how they were taken into account when implementing the system.

- ***Authentication***: To secure the system, an authentication system was implemented in the mobile application, so that only register users can access both the mobile app and the server.

- ***Authenticity***: Achieved by the server's employment of authenticity techniques, confirming that the received requests is cleared to access the desired functionalities.

- ***Confidentiality***: To secure critical user data, an encryption system was implemented in the server, so that the said information is encrypted before being stored in the database.

- ***Usability***: The mobile application was designed having in mind the usability requirements, by implementing an appealing, clear and intuitive interface for the user.

- ***Reliability***: To make a reliable, error-tolerant system, the *Redux* technology was used, so that the mobile app's state information stays consistent and thus, error-free.

- ***Performance***: By employing *Redux* and *AsyncStorage* in the mobile app, the system's performance improves, due to having less need of making server calls as frequently and keep the user waiting.

- ***Portability***: By developing the mobile app using *Expo React Native*, the system was automatically available for Android and iOS.

- ***Testability***: In order to test the devised system, an entire chapter, Evaluation, will be dedicated to that.

- ***Maintainability***: The system overall presents a great degree of maintainability, expressed by the language support technique, which allows for an easy introduction of new languages and also migrations which ease the process of modifying the data schema.

- ***Localizability***: The system was designed, both in the mobile app and the server, to support multiple languages and making it easy to add other ones on demand.

6.7 Tests

This section is dedicated to the testing and validation of the developed software. It's a very important phase in the software development life cycle since it makes sure that the software fulfills the established requirements.

6.7.1 Unit Tests

Unit testing is where small parts of an application, units, are tested, in order to reassure that the functions work as expected. The goal here is to find unspotted implementation flaws inside each individual unit.

```
[TestMethod]
0 references | 0 changes | 0 authors, 0 changes | 0 exceptions
public void GetSlope()
{
    // Arrange
    LinearBestFit linearBestFit = new LinearBestFit();
    var expectedResults = 2;

    var xyPoints = new List<XYPoint>
    {
        new XYPoint { X = 1, Y = 2 },
        new XYPoint { X = 2, Y = 4 },
    };

    // Act
    var line = LinearBestFit.GenerateLinearBestFit(xyPoints);
    var result = LinearBestFit.GetSlope(line);

    // Assert
    Assert.AreEqual(expectedResults, result);
}
```

Figure 120 – Unit Test for the *GetSlope* method

Figure 120 depicts a unit test done to one of the methods of the class responsible for calculating the line of best fit. Specifically, the represented method calculates its slope.

In the Appendix I the rest of the unit tests are presented.

6.7.2 Integration Tests

With the purpose of extending the unit tests, integration tests were built. In these tests, two already tested components are put together and tested as a whole. This helps to find bugs that couldn't be covered within unit tests (e.g. an instance of a class receiving a null instance of another one).

Figure 121 depicts the integration test relative to the method responsible for fetching information from the database relative to the list of Programs.

```
[TestInitialize]
0 references | 0 changes | 0 authors, 0 changes | 0 exceptions
public void Initialize()
{
    mock = new Mock<IPlanRepository>();

    programs = new List<Program>
    {
        new Program { Code = "1", CreatedType = CreatedType.Template, State = StateType.Inactive,
            Level = ExperienceLevelType.Intermediate, IsRepeatable = true,
            MainBodyAreaId = BodyAreaType.Full, ApplicationUserId = "1"},

        new Program { Code = "2", CreatedType = CreatedType.Template, State = StateType.Inactive,
            Level = ExperienceLevelType.Intermediate, IsRepeatable = true,
            MainBodyAreaId = BodyAreaType.Full, ApplicationUserId = "1" },

        new Program { Code = "3", CreatedType = CreatedType.Template, State = StateType.Inactive,
            Level = ExperienceLevelType.Advanced, IsRepeatable = true,
            MainBodyAreaId = BodyAreaType.Upper, ApplicationUserId = "1" }
    };

    mock.Setup(m => m.GetPrograms()).ReturnsAsync(programs);
}

[TestMethod]
0 references | 0 changes | 0 authors, 0 changes | 0 exceptions
public async Task GetProgramsTestAsync()
{
    var expectedResult = programs;
    ProgramsController controller = new ProgramsController(mock.Object);

    // Act
    List<ProgramDTO> result = await controller.GetPrograms();

    // Assert
    CollectionAssert.AreEquivalent(result, expectedResult);
}
```

Figure 121 – Integration Test for the *GetPrograms* method

In the Appendix J the rest of the integration tests are presented.

6.7.3 System Tests

System tests are a core part of the testing phase because they constitute the process of testing the whole integrated system to see if it meets the specified criteria. The system tests were done by combining multiple use cases and/or functionalities, executing them, and verify if they meet the criteria they were supposed to. Table 8 depicts the system test for a normal flow when creating a new Program and then searching for its details.

Table 8 – System test nº1

Description	The user creates a Program, opens the Programs list and views its details.
Result	Success.

In the Appendix K the rest of the system tests are presented.

6.7.4 Acceptance Tests

Acceptance tests are conducted to determine if the requirements of a specific functionality (use case) are being met or not. Acceptance tests are *black box* system tests. Each acceptance test represents some expected result from the system. The end user is the responsible for verifying if a use case is working properly.

Like so, the acceptance tests were built with the goal to ensure every requirement, translated in use cases, is met. Table 9 depicts the acceptance test made to the use case relative to viewing the templates for Plans/Programs.

Table 9 – Acceptance Test for Use Case 2 (US02)

Tested Use Case	US02: View training Plan/Program templates
Expected Result	The correct Plan/Program template list is shown.
Result	Success.

In the Appendix L the rest of the acceptance tests are presented.

7 Evaluation

The Evaluation chapter will be dedicated to the exposition of crucial information used to evaluate the designed solution. First, the metrics that will be used to evaluate are presented and described, then, the different hypotheses are formulated and bestowed, and finally, the evaluation methodology is presented. After, the results and experimental analysis are bestowed, which offer different analysis on the results.

7.1 Metrics

Metrics are very important because they "help capture a business goal into a quantitative target […] ." [86] This means that first, the business goal needs to be defined in order to capture metrics to evaluate it. In order to do so, the system was divided into three different sections – front end, back end and personal virtual assistant.

The back end is the part of the system responsible for handling the business logic and the most important part to evaluate is the one that includes the recommender systems. There are two different recommenders – intelligent and conditional. They are different in terms of implementation, algorithms used, and also in the way they are evaluated. Since the Conditional Recommender is based on traceable and understandable conditional decisions (*if* statements), the way to best evaluate it would be by its execution time and inquiry of satisfaction relative to the recommendations' quality. As such, its metrics are:

- Execution time of recommendations.
- Satisfaction relative to the recommendation's precision.

However, because the Intelligent Recommender is based on numerous and hard to trace variables, its evaluation should be more objective, in terms of numbers, and not susceptible to subjective opinions. Also, given that, as it was already disclosed, the context of the information used in the said system does align with the project's, it would be erroneous to try and get satisfaction inquiries on things that are not contextualized. As such, the used metrics are:

- Individual technique's recommendation error.

The aforementioned metric refers to the error that is associated with each one of the used recommendation techniques.

On the other hand, the front end can be quantitively evaluated through the inquiry of satisfaction from the interface's intuitiveness. The front end, composed by the mobile application, serves as a link between the back end and the user, presenting and receiving information in, hopefully, the most intuitive and pleasant manner, and as such, it should be evaluated on that. Thus, its metric is as follows:

- Satisfaction relative to the overall use of the application.

Lastly, the personal virtual assistance job is to aid and assist the users and so, it should be quantitively evaluated through the inquiry of its helpfulness. Thus, its metric is the following:

- Satisfaction relative to the helpfulness of the information.

7.2 Hypotheses

Having all the different metrics formulated, it's now required to produce the hypotheses that will be used to evaluate them.

For simplification and organization purposes, all the satisfaction inquiries were coupled into a single hypothesis and so, two were built. The first one is relative to the user's satisfaction (Conditional Recommender's precision, overall use of the mobile app and helpfulness of the PVA's information), presented next.

H0: User satisfaction

H1: Satisfaction inquiries possess a satisfaction rate above or equal to 4

H2: Satisfaction inquiries do not possess a satisfaction rate above or equal to 4

The second hypothesis is relative to the execution time of the conditional recommender's algorithm, presented after.

H0: Execution time

H1: The system takes less than 2 seconds to compute recommendations

H2: The system does not take less than 2 seconds to compute recommendations

The defined threshold for the execution time metric is based on the assumption that that is the maximum acceptable amount of time to wait for the type of request.

The third hypothesis refers to the error associated with the recommendations of each one of the used techniques in the intelligent recommender system.

H0: Error

H1: Each technique has an error inferior than 1.5

H2: Each technique has an error that is not inferior than 1.5

The defined threshold for the error metric is based on the assumption that that is the maximum acceptable value of a deviation error for a given recommender model.

7.3 Methodology

With the metrics and hypotheses already defined, it is now crucial to present the different chosen evaluation methodologies.

Table 10 – Evaluation Methodologies

Metric	**Methodology**
User satisfaction	Inquiry of satisfaction and group testing
Execution time	Unit testing
Error	Unit testing

Table 10 contains the summarized information relative to the definition of methodologies for each established metric. For the satisfaction-related metrics, it's important to evaluate the satisfaction of users on the different covered topics (recommendations' precision, overall use of the applications and helpfulness of the PVA's information), so a satisfaction inquiry is required. Plus, it is also important to evaluate the quality of the acquired knowledge and to do so, the group testing methodology is chosen.

Moreover, the inquiry to evaluate user satisfaction will be conducted using, for each question, the classification depicted in the following Table 11.

Table 11 – Satisfaction Inquiry Scale's Description

Scale	**Description**
1	Completely disagree
2	Disagree
3	Neutral
4	Agree
5	Completely agree

In addition, it is important to mention that, to acquire a useful and educated satisfaction inquiry response, an expert in the fitness field will be used. Instead of asking different people to use the Conditional Recommender system and then answer a satisfaction inquiry, an expert shall be used, so that a more educated evaluation of the system can be conducted, with their credentials as substantiation.

To do so, the expert's own data will be used as input to the Conditional Recommender – preferences and history. Then, a recommendation will be computed and offered to them. Both the recommendation and the whole recommender process will be provided to the expert, giving them enough data to answer the satisfaction inquiry.

Regarding the execution time metric, since it refers to duration of algorithm completion, the used methodology will be unit tests. The tests will time the durations for several generated recommendations and then take the average of them all. The resulting value will be used to evaluate the execution time of the recommendation system, as desired.

Finally, the error metric will be evaluated through unit testing, where the different techniques' error is to be calculated, using the *MAE* and *RMSE* which are two of the most common metrics used to measure accuracy. Given that there's no consensus on which one is the best, both were employed, and their results will be presented individually.

The *RMSE* was already discussed in earlier sections, and the *MAE* [181] as the name says, *Mean Absolute Error*, it is the average of all the absolute errors. The absolute error is simply the difference between the actual value and the predicted one. As such, the formula for this metric is as follows [181].

$$mae = \frac{\sum_{i=1}^{n} abs(y_i - \lambda(x_i))}{n} \quad (14)$$

Also, it is important to mention that, to test the techniques, only 90% of the input data will be used, and the remaining 10% will be dedicated only for testing purposes. This is because it is not recommended to use data which was used for training the model to also test it.

7.4 Results Analysis

Following the establishment of the metrics, hypotheses, and methodologies, in this section, the acquired results will be analyzed. The section is divided by the analyzed entities: Mobile Application, Personal Virtual Assistant and Conditional and Intelligent Recommender.

7.4.1 Mobile Application & Personal Virtual Assistant

As predicted, a satisfaction inquiry for the Mobile Application and the Personal Virtual Assistant was needed, and, for simplicity terms, both were joined in a single one. That way, the people who used the Mobile Application also used the PVA and then evaluated both at the same time.

The devised inquiry contained 12 questions for the Mobile Application and 4 for the Personal Virtual Assistant, all to be answered with the classification scale on Table 11. The inquiry was given to 50 different people, with only 20 received answers, 40% of initial population and it can be found in APPENDIX B.

In the following Table 12 the acquired data for the Mobile Application satisfaction inquiry is summarized, in the form of response percentage for each question, helping to visualize the distribution of responses.

Table 12 – Response percentage for each question from the Mobile Application's inquiry

	1	**2**	**3**	**4**	**5**	**Average**
Q1	0%	0%	0%	25%	75%	4.75
Q2	0%	0%	0%	40%	60%	4.6
Q3	0%	0%	10%	55%	35%	4.25
Q4	0%	0%	15%	65%	20%	4.05
Q5	0%	0%	0%	15%	85%	4.85
Q6	0%	0%	5%	55%	40%	4.35
Q7	0%	0%	10%	75%	15%	4.05
Q8	0%	0%	0%	40%	60%	4.6
Q9	0%	0%	10%	50%	40%	4.3
Q10	0%	0%	0%	45%	55%	4.55
Q11	0%	0%	5%	65%	30%	4.25
Q12	0%	0%	0%	45%	55%	4.55

Over 95% of the distribution of responses is between 4/5 and 5/5, which is very positive. Also, the worse classification for any response was a 3/5, with the worse occurrence being of 15%, in question 4. Plus, the average classification for all the questions is approximately 4.4/5, which passes the defined hypothesis of being higher or equal than 4/5. This is a clear indication that the application was very well received by the testing users.

The question with the most percentage of perfect classification is the fifth one which is relative to the registry and login process speed, with 85% of people rating it as a 5/5, with the remaining 15% rating with a 4/5, having an average rating of 4.85/5, followed by question number one, relative to the application's overall design and interface, with 75% perfect ratings and the remainder 25% being a 4/5 and an average rating of 4.75/5.

The question with the most amount of 3 out of 5 ratings was question number 4, with 15% of answers, followed by questions 3, 7 and 9, with 10% of answers. It's not very relevant to analyze and scrutinize these answers for being the worse from the entire inquiry because they're not badly rated whatsoever, given that they still average 4/5, for the questions 4 and 7, and 4.3/5, for questions 3 and 9, which is obviously still very good.

The remainder questions sit in the middle of the pack in terms of average ratings, having an average rating between 4.3/5 and 4.6/5, which is still excellent.

Relative to the Personal Virtual Assistant's satisfaction inquiry, the next Table 13 contains the summarization of the acquired data, in the form of response percentage for each question.

Table 13 - Response percentage for each question from the PVA's inquiry

	1	**2**	**3**	**4**	**5**	**Average**
Q1	0%	0%	0%	20%	80%	4.8
Q2	0%	0%	10%	55%	35%	4.25
Q3	0%	0%	0%	80%	20%	4.2
Q4	0%	0%	0%	5%	95%	4.95

From the results of the PVA's satisfaction inquiry, it can be said that 97.5% of the responses are between 4/5 and 5/5 with the average rating for all questions being approximately 4.4/5, which is the same as the Mobile Application's, which passes its defined hypothesis of being above or equal to 4/5. The only response with a rating of 3/5 is the second one, still having a 4.25/5 average rating.

The questions with the most perfect classifications and also the highest average rating are questions 4 and 1, with 4.95/5 and 4.8/5 average rating, respectively. The fourth question, which has an almost perfect rating, is the one where users were questioned if the PVA was a great asset to the Mobile Application, which is an excellent feedback to receive from end-users.

Even the lowest average rating questions still have an approximate 4.2/5 rating, which is still very good. Regardless, since these refer to the PVA's ability to understand and interpret what was said, it wouldn't hurt to try and improve the prediction system so that the answers would be even more adequate.

7.4.2 Conditional Recommender

As it was established, the Conditional Recommender is to be evaluated based on two metrics – its execution time, and a satisfaction inquiry from an expert on the field.

Firstly, for the measurement and evaluation of the Conditional Recommender's execution time, the entire algorithm was executed whilst recording the time it took to finish. Using the *StopWatch* [182] class, from *Microsoft*, the timer was started before the execution of the algorithm, stopped at the end and then the result was written to a csv file. In order to eliminate anomalies and normalize the data, this was tested a total of 30 times.

To test the algorithm and simulate it with an enormous amount of data, 250 thousand training Workouts samples were fed to the algorithm simulating the database's Workouts, joined with 200 training Workouts simulating the user's history.

As a small recap, the Conditional Recommender algorithm receives as input the list of Workouts that can be recommended to the user, which is the list of Workouts existent in the database, the user's history, which is the list of the user's past Workouts, and the user's preferences such as their experience level, preferred training types, and other preferences whose scale is negligible comparing with the potential magnitude that the database workouts and the user's history have in the execution time.

Even though the volume of input data was very extensive, the algorithms simplicity allowed for it to be swift, with an average execution time of about 870 milliseconds, which is less than 90% of a second. Needlessly to say that it is incredibly fast to compute a response, even with extreme conditions in terms of data size, which is good news for the Performance aspect of the Non-Functional Requirements.

In terms of the defined hypothesis of having an execution time lower than 2 seconds, it can be said that the algorithm was a success, given that the real value was approximately 57% lower than the defined threshold.

As a final note, regarding the evaluation of the execution time, the acquired data in the form of a csv file can be found in the APPENDIX E.

Secondly, for the evaluation of the Conditional Recommender through a satisfaction inquiry of a field expert, a formal request was made to the said expert, seeking for their personal information to be fed into the system as an official test to its quality.

As such, the information regarding the expert's recent training history and personal preferences (i.e. experience level, training types, goals, etc.) was sent as input to the algorithm. Then, the whole recommender process, already described in its respective section, was documented and thoroughly described in a formal document, following the *modus operandi* of the algorithm up until the very last selection. In other words, the whole process of filtering the initial list of

Workouts to find a final adequate recommendation was detailed, specifically detailing what is being done, how is it being done and what is the produced answer of each phase.

The produced document was then presented to the expert and can be found int APPENDIX C, who then reviewed it and based on not only the final recommendation but also on the whole process, answered a satisfaction inquiry, which can be found in APPENDIX D.

This inquiry's results are exhibited in the following Table 14, where it can be immediately observed that one question was poorly rated. This question refers to the efficiency of the evaluation of the volume and intensity and it is considered to be the number priority when discussing improvements in the respective system. Moreover, another question that had a rather low classification was the last one, referent to the application of the filtering variables. Because the adequacy of the said variables is confirmed by the classification with the maximum rating, in the fifth question, it can be assumed that the issue lies in the application of the already obtained variables and not so much on their definition.

All other questions were classified with a 4/5 rating, which results in a 3.7/5 average rating. This final rating means that the defined hypothesis was not fulfilled, since it was required to possess a rating higher than 4/5. This is a clear indication that there's an evident flaw in the devised system that should be addressed.

Table 14 – Expert's satisfaction inquiry results

Question	**Classification [1-5]**
Q1: The recommendations are great.	4
Q2: The recommendations are adjusted to my profile.	4
Q3: The volume and intensity evaluation is efficient.	2
Q4: The volume and intensity prediction is efficient.	4
Q5: The variables used for filtering are adequate.	5
Q6: The filtering of the variables is efficient.	3

Having presented all the results analysis referent to the Conditional Recommender, in a following section, named Experimental Analysis, the exploration of the algorithm's flaws, its intricacies and improvement suggestions will take place.

7.4.3 Intelligent Recommender

The final analysis has to do with the most complex and difficult to evaluate component – the Intelligent Recommender. Its analysis, as it was already defined, will focus on the error of the produced recommendations, for each one of the used techniques, using two different error metrics – *MAE* and *RMSE*.

Firstly, the to be tested techniques were established, given that there are three major techniques implemented: *Matrix Factorization (MF), User-Based Collaborative Filtering (UB)* and *Item-Based Collaborative Filtering (IB)*.

Even though only three techniques were implemented, in order to add some customization, some variations of them were also devised. Firstly, a combination of all these techniques, using a *Hybrid* approach was taken. Then, in terms of the UB, 4 variations were also added, one for each of the Comparers. As such, the techniques used for testing were as depicted in Table 15.

Table 15 – Error Analyzed Techniques

ID	Technique Composition
1	MF
2	UB with Pearson's Correlation (PC)
3	UB with Cosine Similarity (CS)
4	UB with Co-rated Cosine Similarity (CR-CS)
5	UB with Root Mean Squared Error (RMSE)
6	IB
7	MF + UB with CR-CS
8	MF + IB
9	MF + UB with CR-CS + IB
10	UB with CR-CS + IB

With all the to be used techniques defined, a recommender model was created for each one, using the *Train* method, which was already described. Afterwards, the model's error was calculated using a specific class, *RecommenderMeasures*, containing two methods, for computing the MAE and RMSE errors of a given recommender model. These methods receive as argument the recommender model, and the data used for testing (10% of the original dataset), then they produce recommendations using the model's *getRating* method, and compare the result with the actual rating value (from the original *Ratings Matrix*). Each of the methods, for MAE and RMSE will then produce an error associated with the built model.

These errors were calculated a total of 30 times for each technique and the average of the results for each one can be found in the next Table 16, where the average and the difference between the two error metrics can also be found. The full results can be found in APPENDIX F.

Table 16 – Recommender technique's error

ID	MAE	RMSE	Difference	Average
1	0.77	1.01	0.24	0.89
2	3.10	3.39	0.29	3.24
3	2.04	2.33	0.29	2.18
4	1.80	2.15	0.35	1.98
5	3.16	3.37	0.21	3.26
6	0.91	1.11	0.20	1.01
7	1.56	1.78	0.23	1.67
8	0.78	0.98	0.20	0.88
9	1.45	1.61	0.16	1.53
10	1.79	1.95	0.16	1.87

Before delving the technique's intricacies, it is important to notice that the RMSE is always larger than the MAE, and the difference between the two is relatively speaking very consistent, with an average of 0.23. This difference can be explained by a paper by *T. Chai* and *R. R. Draxler* [183] which states that by definition, the RMSE can never be as small as the MAE, due to the RMSE allowing for a complete reconstruction of the error set, given enough data, and the MAE only accurately recreating 80% of it. This might be why most cost functions used in machine learning systems avoid using MAE and rather use RMSE or variations. [184] Also, RMSE tends to disproportionately penalize large errors since the error is then squared, which directly affects the produced error. On the other hand, MAE does not have any bias towards extreme values, which makes it more prone to being lower in value. [184]

Regardless, by quick reading the previous table, it is evident that the lowest average error is from technique 1 and 8 which are the MF and MF + IB techniques. This suggests that using MF as a standalone technique is the most beneficial course of action, only being surpassed by combining it with IB, even though the errors are pretty much identical statically speaking.

The two worse performing techniques are the technique 2 and 5 which unironically are two variations of the UB technique, using the *Pearson's Correlation* and RMSE comparers, respectively. This might suggest that these are not good comparing techniques. There are other two variations, using the *Cosine Similarity* and the *Co-rated Cosine Similarity*, technique 3 and

4 respectively. The difference between these two and the two best ones is of over 55%, which indicates that using the *Cosine* or *Co-rated Cosine* is far superior. Regardless, comparing it with the best performing techniques there's still a 236% difference, which is huge and also suggests that using UB, even with the best comparers is very inferior to using MF or even MF with IB.

Moreover, given that the UB with the *Co-rated Cosine Similarity* is the variation of the UB with the lowest error, it was the one used in all other *hybrid* approaches.

Another technique which appeared to possess a small error, only 14% bigger than of the smallest ones, is the technique 6, IB, which makes sense, given that when combined with MF it also produces an even smaller error. It is curious that even though the IB technique's error is 14% larger than the MF's, when combining the two, the MF's error is maintained, which might suggest that the two work well together, for the given dataset.

The remainder techniques, 7,9 and 10, sit in the middle of the pack in terms of their error. These are all hybrid variations containing the worse performing technique, UB, which might be the reason why they sit in the middle of the list - because they're coupled with the best performing techniques, they increase their otherwise low error.

From the analyzed data, for the given dataset and context, the following conclusions can be made:

- The MF technique is the best performing one, individually speaking.
- The IB technique performs very well, taking into account that it is not an intelligent technique, when faced against an intelligent and superior one, MF.
- Coupling MF with IB maintains the low MF error and might be productive.
- UB is a low performing technique and unlike IB, coupling with other better performing techniques will just increase their error.

As a final statement, it is important to mention that only three techniques passed the devised hypothesis of the error being lower than 1.5, which are the techniques number 1, 6 and 8, which are MF, IB and a combination of both, respectively. Two more techniques, 7 and 9, were very close to passing the hypothesis but fortunately they did not, given that they were just weaker variations of the ones that passed, as it was already explained.

7.5 Experimental Analysis

Even though a result analysis was already conducted, there are still some investigation that can be conducted, in order to find different deviations of results by gradually changing certain variables and observe how the object being analyzed behaves under those circumstances, which constitutes an experimental analysis. This analysis can be viewed as investigation for future work.

For example, it would be very interesting to find how the Intelligent Recommender's error would change by raising variables such as the number of iterations or number of used features, so that the system can be better understood and improved.

Moreover, regarding the Conditional Recommender, since the error is not a metric being evaluated and analyzed, one can investigate through a different route, such as trying to gather the comprehensive feedback of the expert as a way of understanding more concisely where the algorithm needs improvement and where it is needs not.

Consequently, the next sub-sections will be dedicated to the exploring of these topics, that can be summed as such:

- Intelligent Recommender:
    - Error variation with the increase of *latent features* for the *Matrix Factorization* technique.
    - Error variation with the increase of iterations (*epochs*) for the *Matrix Factorization* technique.
    - Error variation with the increase of closest neighbors for the *Pearson's Correlation* in the *User-Based Collaborative Filtering* technique.
- Conditional Recommender:
    - Expert's feedback on the algorithms overall flow.
    - Expert's feedback on the individual filtering phases.

7.5.1 Intelligent Recommender

As it was explained in previous sections, the *Matrix Factorization* technique uses *latent features* to find underlying attributes that are expressed through user tastes and item's preferability. As such, the number of features is a debatable and honestly subjective topic of discussion. It depends not only on the data's context, but also on its magnitude. If there's too little data, having a certain number of features that are above the number of necessary ones to fit the model perfectly, might cause the it to overfit, which is not desirable at all.

To find if a model is overfit, one must measure the training error – error associated with the training phase, as the one presented in the Result Analysis section – and compare it with the

measured testing error – error associated with the testing phase, where the model is already trained. Then, four different possibilities can occur:

1. Training and testing error are high.
2. Training error is low, testing error is high.
3. Training error is low, testing error is slightly higher.
4. Training error is high, testing error is low.

From an intuitive perspective, these possibilities reflect on the model's fit. If the training error is high, it means that the model wasn't well trained, and therefore, if the testing error, for the same model, is also high – possibility 1 – it makes sense to infer that the model is underfit. If, on the other hand, the testing error is low – possibility 4 – the model is impossible to evaluate, because there's no reason for an undertrained model to make good predictions (low error).

Moreover, if the training error is low, the model can either be well fit or overfit. If, for this model (low training error), the testing error is high, it means that the model is overfit – possibility 2. This is why it is important to measure both the training and testing error, so that one is not misguided to think the model is well trained just because it had a low error during training, as that can be due to it being overfit to the data, which will then result in a high error with untrained, and therefore unfit, data, since it is not fit to generalize and make good predictions. What is considered to be a good fit model is when besides having a low training error, the testing error is also low, though usually a bit higher than the training one – possibility 3. That intuitively tells us that after producing a low training error, the model's validity can be verified by having an also low testing error which means that the said training error was low enough to make the model well fit, but not too low to overfit it.

With all these assumptions in mind, the model was trained using an increasingly higher number of *latent features* and the training and testing error was measured, to find what the most optimal range of features would be for the given dataset. The used error metric was the RMSE. The results can be summed up in the following Table 17 and can be viewed in their entirety in APPENDIX G.

Table 17 – Training and Testing error variation with the number of *latent features*

Features	Training Error	Testing Error	Difference
1	0.76	0.95	0.19
5	0.70	0.98	0.28
10	0.63	0.99	0.36
15	0.58	1.00	0.42
20	0.54	1.01	0.47
30	0.48	1.03	0.55
40	0.43	1.05	0.61

As observable in the previous table, the predicted conclusion is easily verifiable: with the increase of *latent features* comes an increase of the difference between the training and testing errors, justified by the lowering of the training error and increasing of the testing error, which unequivocally confirms the overfitting of the model, which can also be observed in Figure 122. The lowest difference occurs with only one feature, where the testing error is slightly higher than the training error, which might suggest one of two things: either one feature is the most optimal number for the given dataset, or the dataset is so small, relatively speaking, that it's not possible to create a good accurate model. This impossibility is due to the fact that the model cannot have more than one feature without running into overfitting issues, and one feature is not nearly enough to make a good fit model.

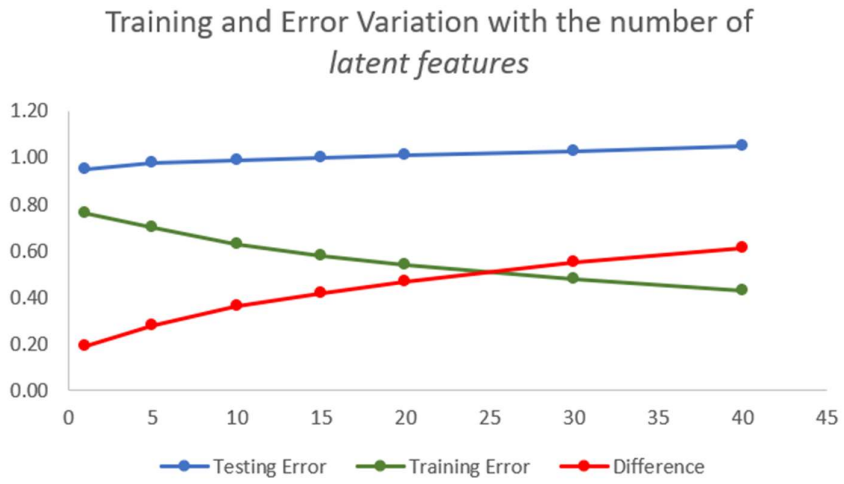

Figure 122 – Graph of the training and testing variation with the number of *latent features*

To further investigate how the model behaves and how to manipulate the error in a different way rather than changing the number of features, as defined previously, the variation of the training error will be analyzed through measuring it whilst increasing the number of iterations, or *epochs*, that the model will go through before finish training. This investigation can be found in APPENDIX H.

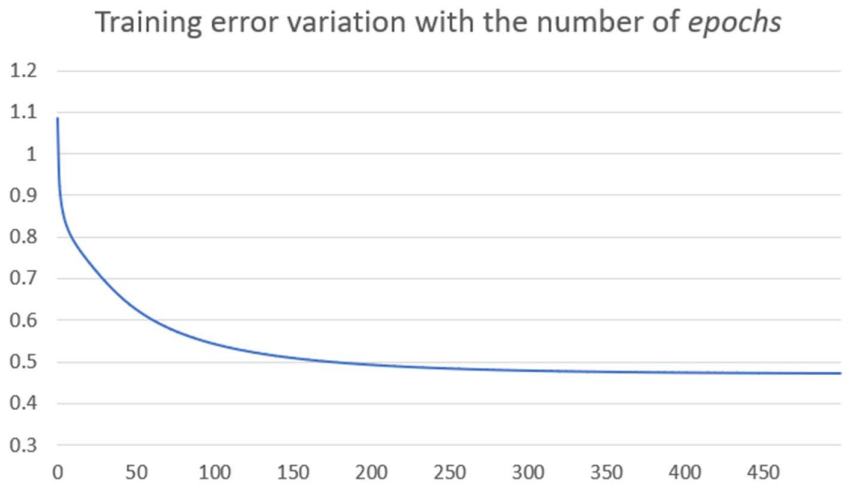

Figure 123 – Graph of the training error variation with the number of *epochs*

By observing Figure 123 it becomes evident that the graph follows an exponential-like curvature, rapidly decreasing in the first iterations and then varying less and less the more *epochs* pass through. This is because it is in the first iterations where the model suffers the most extreme changes. These changes refer to the model's learning process, as described previously, where the model tries to approximate the most optimal value for a given cost function. In this case, the model uses iterations as a "stop term", and suchlike the *latent features*, if there are too many iterations, the model might suffer from overfitting. As such, it should stop after the most aggressive changes were made, so that it has a low error and is well-fit, but before the error reaches a plateau, to prevent overfit.

In the Figure 123's graph, it is evident that stopping the algorithm at 50 *epochs* would not be appropriate because the model will still significantly decrease its error. At 200 *epochs* the error reaches a value that is close to its lowest and at *300* it finally plateaus, so there's no more reason to keep iterating and forcing the model to learn and risk overfitting it. Specifically, any value between 200-300 *epochs* would be debatably optimal for this particular case.

Finally, another important variation to be tested refers to the number of neighbors used in the *User-Based Collaborative Filtering* technique, and more specifically, in the used neighbor-comparing methodology which is the *Person's Correlation*. The reason for using this comparer and not any other was completely random. The idea is not to evaluate the technique in itself, but the changes that varying the number of neighbors produces. If there's the need to further investigate for each one, this exploration serves as guideline.

Either way, this was tested by measuring the *MAE* and *RMSE* error associated with a UB-CF with *Person's Correlation* model, with an increasing number of neighbors, and then analyzing the subjacent impact.

Table 18 - Error variation with the number of neighbors

Neighbors	**MAE**	**RMSE**
1	3.50	3.66
5	3.35	3.59
10	3.21	3.46
15	3.15	3.42
20	3.10	3.39
30	2.84	3.24
40	2.78	3.16

In the previous Table 18, the *MAE* and *RMSE* error for each number of neighbors is depicted. Here, it is evident that with the increase of neighbors, the error diminishes, which makes sense, given that searching for a broader number of users will increase the odds of finding more adequate ones.

The graph from Figure 124 helps to illustrate the decrease in the error with the increase of neighbors. It is also evident that the *RMSE* error is always considerably higher than the *MAE*'s. The reason why was already covered before, but a more curious finding has to do with the fact that, with the increase of neighbors, the difference between the two errors also increases slightly. This variance could be overlooked, but it does seem to be significant, and the reason for it may be explained because, as stated before, the *RMSE* is more susceptible to large errors, and with an increasingly number of neighbors, even though the likeliness of finding a more adequate near neighbor increases, the probability of finding bigger deviations from the acquired neighbors also increases.

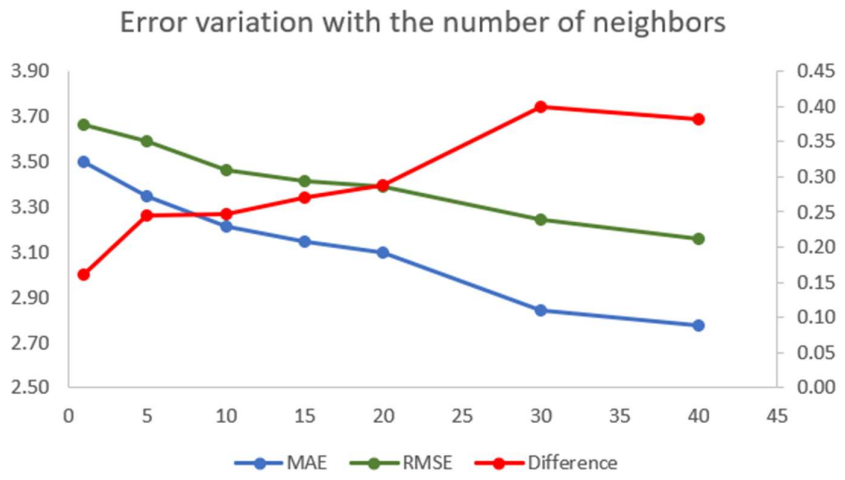

Figure 124 – Graph of the error variation with the number of neighbors

Even though the overall conclusion is that the error continuously decreases with the number of neighbors, the unpredicted finding relative to the increase in the difference between the two types of errors was captivating and rather insightful.

7.5.2 Conditional Recommender

Having presented the satisfaction inquiry results in the previous Results Analysis section, there's still crucial and noteworthy information acquired from the interaction with the expert. A more in-depth feedback was received, and this section will be dedicated to its exposition with the purpose of attaining new knowledge so that improvements on the Conditional Recommender can be pursued in the most pivotal areas, as future work.

As defined, the acquired feedback can be divided in two sections: overall flow of the algorithm and individual critique for each filtering phase.

The overall flow of the algorithm, in the expert's opinion, makes perfect sense and it is a good approach to find adequate workouts to recommend. The fact that the recommendations are based on existing workouts adds credibility to the recommendations, since there's no need to invent and create workouts from scratch, which could lead to more debatable and complex decisions regarding the recommendations.

The stage where the algorithm really shines is in the prediction of the intensity and volume values. Here, by analyzing previous instances, patterns are found and identified so that an accurate educated guess on the predicted value for the intensity/volume can be made. Even though some patterns cannot be captured using the line of best fit, it is most definitely applicable to most cases where there aren't extreme variations in an almost random fashion.

As such, the overall flow can be considered a success and part of the reason why the algorithm works rather accurately.

In terms of critiquing individual sections of the algorithm, the one that demanded the most attention and analysis was the *volume and intensity evaluation* phase. In this phase, the volume and intensity of each training workout is evaluated and given a number correspondent to the number of repetitions and its intensity value (represented by a number on a scale of 0 to 3), respectively.

This phase, in the expert's opinion was the one that had the most deficiencies, having an imprecise and vague concept of evaluating volume and even more so on the intensity aspect. Specifically, the volume can be evaluated through the number of repetitions for a given training session, but a workout with more repetitions than other might not necessarily imply it has higher volume, because there are other ways of measuring volume, suchlike the duration. This becomes especially imprecise when dealing with both repetition and duration-based exercises in the same session, which makes it harder to level the two a find a way to compare them.

In terms of the intensity evaluation flaws, similar to the volume's, there are many ways to evaluate intensity such as RPE and percentage, as it was used in the algorithm and covered before, but also through the lifted weight, heart rate, and other metrics. This imposes a difficulty when dealing with more than one metric, for the same reason covered in the volume – because it becomes hard to compare different measures, especially if they have a high appearance variance. This predicament is notably troublesome when dealing with intensity because there are more possible metrics and also because they're particularly arduous to make comparisons between.

Another issue in this *volume and intensity evaluation* phase is the fact that they, the volume and intensity, are being generalized for the entire workout which can lead to increased imprecision, because the intensity and volume of that session is basically being averaged by all the exercises and that may not be representative. As such, a suggested approach would be to evaluate each exercise individually and make assumptions on the weight it should have on the session's overall value, based on aspects such as its difficulty, and also its duration, number of sets and reps, and intensity's ratio in the entire workout – an exercise that takes up most of the workout to complete should have a bigger weight on its evaluation.

Even though the previously described filtering phase was the one with the most flaws, there were also other small aspects in different ones that are noteworthy and should be documented for the sake of future work improvements.

Firstly, it was described that when a given filter returns no elements, for example when trying to find the untrained muscle groups and all the muscle groups were already trained, the algorithm simply resets the specific filter. In the given example's case, all the muscle groups from the user's preferences would be considered for filtering. This is a decent approach, but not an exceptional one.

From debating with the expert, it became clear that when encountering a case like the one described, the best course of action should not be to simply reset the preference, but to thoroughly evaluate past occurrences. As an example, if the user preferences indicate they want to train chest and arms and both of these muscle groups are included at least once in their past workouts, the proportion for each one should then be examined. Even though both were trained, if the chest appears in 90% of the workouts and arms in only 25%, then, a training workout containing arms might be of the user's best interest to be recommended.

Obviously, this is a very unsophisticated description of a possible improved procedure, but it just goes to show that the algorithm can be greatly improved by implementing more refined and individual approaches to each one of its stages.

8 Conclusion

The main goal procured from the elaboration of this dissertation was to develop a personalized monitoring and planning fitness system, in conjunction with a fully capable recommender, adapted to the individual users' profile and needs. The key aspect with the envisioned system lies in the individuality that it must offer, allowing users to manage, monitor and acquire data.

Due to its extent and level of complexity, the project's load was divided by two authors, with clearly defined and separate responsibilities, having of course, various elements in common. This dissertation focuses specifically in the training programming aspect of the system and all its subtleties such as modeling, creating and managing training programs.

After introducing the problem, goals, motivation and used methodology, the value analysis was produced, thoroughly contextualizing the project in its scope, presenting the model responsible for identifying opportunities and selecting new ideas, and exhibiting a value proposition. This helped to determine and analyze the value that the project might offer, which was concluded to be very high, due to the pursuing of a system that treats users individually and makes adequate, intelligent recommendations depending on numerous factors which are specific to each one.

In order to find what was already being offered in the market, and what solutions there were to implement a system capable of making recommendations, the applications with the most resemblance to this project, and the most common recommendation methodologies were selected and thoroughly analyzed. It was then concluded that even the most popular applications in the market have lacking areas that can be fulfilled by the current project, and that there's a market hole for what this dissertation is trying to achieve, in terms of individualization and intelligent recommendation. Moreover, the pros and cons of each recommendation technique were laid out, so that later an informed decision can be made regarding which ones to use for the given context.

During the design process, the requirements were engineered, so that the system could hold the desired and envisioned functionalities, through the definition of specific use cases and the establishment of a set of non-functional requirements that should be respected in order to maintain a high level of inherent quality. Moreover, the architecture to be used was

investigated, by designing different ones that responded better in certain situations and worse in others. These were weighed and a final decision of an architecture fully capable of sustaining the engineered requirements was made. The chosen alternative is characterized by having a medium, or *middleman*, between any components and the database and recommender system, encapsulating the responsibilities to a single component, reducing the risk of inconsistency. For the context of the project, it was a far from robust alternative, but one that was fully adapted to everything set forth.

After designing the desired requirements and defining the most adequate architecture, the specific functionalities were implemented. After implementing the use cases, the intelligent recommender was also developed. It was during this implementation that it was decided that a second recommender, a non-intelligent one, was necessary. This need emerged because it was discovered that it would be impossible, in the timeframe of this dissertation, to acquire and test the intelligent recommender with contextually true information.

In other words, due to the impossibility of obtaining a large enough dataset to feed the intelligent recommender to make it viable, as it was required, with information regarding the scope of this project, a new dataset had to be selected, so that the recommender's algorithm integrity could still be tested on the absence of the ability to test it with real applicable data. Regardless, the selected dataset was exquisite and widely used for testing terms across multiple fields of investigation.

It was then decided that a second recommender system, that could apply the project's data, had to be devised. This recommender was based on conditional statements, that found the most adequate recommendation by process of elimination, filtering unsuitable recommendations based on a plethora of variables, individual to each user.

After implementing, the devised system was also tested using unit, integration, system and acceptance tests.

The system was then subjected to a comprehensive evaluation, defining metrics and hypotheses and applying them using predefined methodologies. The results were then analyzed and discussed. In the end, an extra experimental analysis was conducted, discussing aspects that were not covered in the previous one.

A fitness expert was used in the evaluation of the Conditional Recommender, so that its quality could be validated and made credible.

8.1 Accomplished Goals

From the defined goals, it can be said the main one, relative to the implementation of a working system, capable of personalized monitoring and planning allied with a smart recommender system was achieved. Regardless, there were still some lacking areas, all of which can be observed in the next Table 19.

Table 19 – Accomplished Goals

Goal	Degree of Accomplishment
Investigation and analysis of different recommendation techniques.	Fully Accomplished.
Investigation and analysis of different fitness mobile applications.	Fully Accomplished.
Develop a mobile application.	Fully Accomplished.
Develop a web application.	Not Accomplished.
Implement Program-related monitoring and management.	Fully Accomplished.
Integrate a Personal Trainer or Coach in the system.	Not Accomplished.
Develop an Intelligent Recommender.	Accomplished to some extent.
Develop a Personal Virtual Assistant	Fully Accomplished.
Elaborate a study evidencing the utility of the developed system followed by the analysis of its results.	Fully Accomplished.
Fulfill the software development cycle.	Fully Accomplished.

In the previous table, the proposed objectives and their degree of accomplishment are presented. It is important to mention that the only not accomplished goals are the ones referent to the web application and the integration of the Personal Trainer/Coach in the system. These were not considered to be of high priority and due to time restrictions, were left behind. The fact that the web application was not implemented, is the reason why the back office planned during design was also not implemented.

Another important note refers to the development of the Intelligent Recommender. As it was described through the document, due to time restrictions it was impossible to build an intelligent system with applicable data, so data with not contextually relevant had to be used, which is the reason for it to not be considered fully accomplished. Regardless, to compensate, another recommender was implemented, as described previously also.

In sum, it can be concluded that the main goals were implemented with a high degree of success.

In terms of specific functionalities, the next Table 20 illustrates the degree of accomplishment of the individual designed use cases, ordered by their priority.

Table 20 – Accomplished Use Cases

Use Case	Priority	Degree of Accomplishment
US02: View training Plan/Program templates	High	Fully Accomplished.
US03: Manage training Plan/Program	High	Fully Accomplished.
US04: Manage Body Measurements	High	Fully Accomplished.
US06: Recommend training Plan	High	Not Accomplished.
US07: Recommend Workout	High	Fully Accomplished.
US08: View training Logs	High	Fully Accomplished.
US10: View Statistics	High	Fully Accomplished.
US13: Interact with PVA	High	Fully Accomplished.
US15: Change Language	High	Fully Accomplished.
US01: View current training Plan/Program	Medium	Not Accomplished.
US11: Monitor progress	Medium	Not Accomplished.
US16: Change Unit System	Medium	Fully Accomplished.
US17: Manage Available Equipment	Medium	Fully Accomplished.
PT01: Monitor trainees	Medium	Not Accomplished.
US05: Add favorite training Plan/Program	Low	Not Accomplished.
US09: Share training Plan/Program	Low	Not Accomplished.
US12: Interact with PT/Coach	Low	Not Accomplished.
US14: Check social feed	Low	Not Accomplished.
NT01: Monitor clients	Low	Not Accomplished.
AD01: Add predefined training Plan/Program	Low	Not Accomplished.
AD02: Manage scientific articles	Low	Not Accomplished.

As it was presented from the start, not all use cases possessed the same priority, and due to time restrictions, some would have to be prioritized. In the previous table, the implemented and not implemented use cases are depicted, and there it can be seen that the only high priority use case not implemented refers to the recommendation of training Plans. This is because recommendations were only possible by using the Conditional Recommender, and that is only fit to deal with training Workouts.

The use cases with medium priority which were not implemented refer to the viewing of the current training Plan/Program, monitor progress, and monitor trainees. From their priority group, these can be considered the lowest ranked of them, since they're not required to have a working system, they're complementary.

Also, none of the low priority ones were implemented, which makes sense, because if there was no time to implement all of the high nor medium priority use cases, it shouldn't also be enough for the low ones. Regardless, these do not constitute important and critical use cases whatsoever, just good-to-have ones.

8.2 Limitations and Future Work

The current project had, as any, its flaws. Whether due to time restrictions or limitation on the used technologies, it's no lie that the project could be improved in some areas.

Regardless of the fact that the main goal set was accomplished with a relatively high degree of success, there are some limitations that even though do not constitute major flaws in the system, could be amended. These limitations are expressed through future work in the following Table 21, separated by their respective component.

Table 21 – Project's Future Work

Component	**Future Work**
Mobile APP & API	• Implementation of the remainder use cases. • Improve the Mobile App's interface, speed and overall functioning based on the received feedback.
Personal Virtual Assistant	• Add new intents. • Improve the intent prediction system.
Intelligent Recommender System	• Find or build an adequate, applicable dataset. • Investigate and apply *Cross-validation*. • Investigate and apply *Regularizations*. • Investigate new recommendation techniques. • Investigate new comparing techniques. • Investigate the "*latent features* issue"

Conditional Recommender System	• Add additional inputs.
	• Improve the model based on the expert's feedback:
	  ○ Improve the volume/intensity prediction algorithm.
	  ○ Improve filtering technique.
	• Explain recommendation's reasoning.

The project's limitations are expressed in the form of future work, but in all truthfulness, the only real limitation of the project is the fact that the Intelligent Recommender didn't use a dataset adapted to the its context. The other points illustrated in the previous table can be just viewed as future work and not so much as limiting factors suffered by the project.

Regardless, some future work points that deserve to be addressed are the ones presented from the Intelligent Recommender System standpoint. There, apart from the dataset issue, it can be found future work regarding *Cross-validation* and *Regularization*. These were already properly described and justified in the respective sections, but briefly they refer to improvements that the intelligent algorithm can apply in order to improve its recommendations and error prevention. Also, new recommendation and comparing techniques can be also investigated as a way to add variability and improvement on the overall predicting system.

Moreover, an issue that struck the project when evaluating the intelligent system is relative to the fact that it was found that using just one *latent feature* was the most adequate solution for producing recommendations when using the *Matrix Factorization* technique. Since across literature it is pretty common to use multiple, even dozens *latent features*, it should be investigated the reason for this particularity.

Another noteworthy future work point refers to the addition of further inputs to the Conditional Recommender System's algorithm. These can be helpful when trying to find adequate recommendations suchlike providing predetermined filtering options which reduces the number of required filters and adapts better to the user's taste. For instance, the user should be able to input what type of Workout they want, specifically, which will be a filtering stage in itself, limiting the results in their favor. If the user wants a Workout for Legs, it shouldn't matter which muscle groups they still have untrained, so that filtering stage is bypassed and only Leg Workouts will be recommended, which is obviously more in line with what the user desires.

Finally, still in the Conditional Recommender System component, explaining to the user the reasoning behind the choosing of the offered recommendations can be a relevant piece of information to provide, which will strengthen the user's confidence in the algorithm. Saying that a given Workout was recommended because the user hasn't been focusing on specific characteristics such as muscle groups or training types that they defined as desired and that the recommended Workout offers is a very meaningful information.

8.3 Final Appreciation

It was understood from the start that this dissertation would be a challenge. Given that it was extensive enough to support two authors to completely build it, it had to be perfect to work as intended. The fact that a Mobile Application, using never used technology had to be implemented from the ground was also very challenging, since the it took a toll on the planning to learn the new technology, and the same goes for the Intelligent Recommender System. Machine Learning and AI were topics only until then read on articles and big company's headlines. Building and exploring a true intelligent system was a remarkable feat and one that personally opened my eyes to what was truly possible with "simple" algorithms.

From a professional standpoint, dealing with schedules, milestones and collaboration with another author was very rewarding, since it allowed for the development of crucial skills to be used in the future such as time, people, and resource management.

In a more personal level, the dissertation exceeded my expectations, both in the end product as well as in the benefits that will forever translate into my professional career such as teamwork and also general adaptiveness towards the achievement of common goals outside my comfort zone.

To summarize, this project global appreciation is very positive and regardless of its limitations, I personally believe it was very successful and there's a great sentiment of satisfaction in the produced result, both in the full documentation aspect of it and also in the more tangible parts such as the devised Mobile Application and built Recommender Systems.

Bibliography

[1] AGAP, "Missão," [Online]. Available: http://www.portugalactivo.pt/missao. [Accessed 15 2 2019].

[2] Meios e Publicidade, "É este o retrato do sector de ginásios em Portugal," 6 7 2016. [Online]. Available: http://www.meiosepublicidade.pt/2016/06/e-este-o-retrato-do-sector-de-ginasios-em-portugal/. [Accessed 15 2 2019].

[3] A. S. Rojas, "Sport ScienceReview," *I'm super-setting my life! An ethnographic comparative analysis of the growth of the gym market,* pp. 276-299, 20 1 2017.

[4] Deloitte, "European Health & Fitness MarketRepor t 2018," 2018.

[5] "Why is following a workout program so Important?," 1 4 2016. [Online]. Available: https://beethewellness.com/2016/04/01/why-is-following-a-workout-program-so-impotant/. [Accessed 15 2 2019].

[6] C. Cardoso, "Want better results at the gym? A structured program could be the answer," [Online]. Available: https://www.copemanhealthcare.com/resources/want-better-results-gym-structured-program-answer. [Accessed 15 2 2019].

[7] Realbuzz, " How To Monitor Your Fitness Progress At The Gym," [Online]. Available: https://www.realbuzz.com/articles-interests/fitness/article/how-to-monitor-your-fitness-progress-at-the-gym/. [Accessed 15 2 2019].

[8] J. Thomas, "Proven Methods for Prospects Follow-Up in your Gym," 19 9 2016. [Online]. Available: https://www.linkedin.com/pulse/proven-methods-prospects-follow-up-your-gym-jim-thomas. [Accessed 15 2 2019].

[9] Business Dictionary, "value analysis," [Online]. Available: http://www.businessdictionary.com/definition/value-analysis.html. [Accessed 17 2 2019].

[10] Oxford Dictionaries, "Definition of fitness in English," [Online]. Available: https://en.oxforddictionaries.com/definition/fitness. [Accessed 11 2 2019].

[11] R. L. Duyff, Academy of Nutrition and Dietetics Complete Food and Nutrition Guide (5th ed.), Boston: houghton mifflin harcourt, 2017.

[12] L. Chaddock, M. B. Neider, M. W. Voss, J. G. Gaspar and A. F. Kramer, "Do Athletes Excel at Everyday Tasks?," in *Medicine & Science in Sports & Exercise*, 2011, pp. 1920-1926.

[13] centers for disease control and prevention, "About Physical Activity," [Online]. Available: https://www.cdc.gov/physicalactivity/about-physical-activity/index.html. [Accessed 11 2 2019].

[14] U.S. Department of Health and Human Services, Healthy People 2010: Understanding and Improving Health (2nd ed.), Washington: U.S. Government Printing Office, 2000.

[15] Transportation Research Board, Institute of Medicine of the National Academies, Does the Built Environment Influence Physical Activity?, Washington, D.C., 2005.

[16] IHRSA , "The 2017 IHRSA Global Report," 2017. [Online]. Available: https://www.ihrsa.org/publications/the-2017-ihrsa-global-report/. [Accessed 6 2 2019].

[17] IHRSA , "The 2018 IHRSA Global Report," 2018. [Online]. Available: https://www.ihrsa.org/publications/the-2018-ihrsa-global-report/. [Accessed 6 2 2019].

[18] Allegra, "UK Fitness Club Market – Strategic Analysis, Sample Extract, April 2018," 2018.

[19] IBISWorld, "Gym, Health & Fitness Clubs Industry in China," 11 2018. [Online]. Available: https://www.ibisworld.com/industry-trends/international/china-market-research-reports/culture-sports-entertainment/gym-health-fitness-clubs.html. [Accessed 11 2 2019].

[20] IBISWorld, "Gym, Health & Fitness Clubs Industry in the US," 12 2018. [Online]. Available: https://www.ibisworld.com/industry-trends/market-research-reports/arts-entertainment-recreation/gym-health-fitness-clubs.html. [Accessed 11 2 2019].

[21] MarketingToChina, "Fitness industry in China generate 10.3% of annual growth," 17 10 2018. [Online]. Available: https://www.marketingtochina.com/gym-health-fitness-china/. [Accessed 11 2 2019].

[22] AGAP, "Barómetro2017," 2018.

[23] B. Midgley, "The Six Reasons The Fitness Industry Is Booming," 26 9 2018. [Online]. Available: https://www.forbes.com/sites/benmidgley/2018/09/26/the-six-reasons-the-fitness-industry-is-booming/#e6b3f1a506db. [Accessed 6 2 2019].

[24] L. Kesiraju and T. Vogels, "Health & Fitness App Users Are Going the Distance with Record-High Engagement," 7 9 2017. [Online]. Available: https://flurrymobile.tumblr.com/post/165079311062/health-fitness-app-users-are-going-the-distance. [Accessed 12 2 2019].

[25] Kantar Worlpanel, "Nearly 16% of US consumers now own wearables," 25 1 2017. [Online]. Available: https://www.kantarworldpanel.com/global/News/Nearly-16-of-US-Consumers-and-9-in-EU4-Now-Own-Wearables. [Accessed 12 2 2019].

[26] J. Boly, "How To Use The RPE Scale For Strength Training (Plus What The Research Suggests)," 28 12 2018. [Online]. Available: https://barbend.com/how-to-use-rpe-scale-strength-training/. [Accessed 15 2 2019].

[27] J. Boly, "3 Ways to Find Your 1-Rep Max (Beginner, Intermediate, and Advanced)," 10 1 2017. [Online]. Available: https://barbend.com/3-ways-to-find-your-1-rep-max-beginner-intermediate-and-advanced/. [Accessed 15 2 2019].

[28] Journal of Sports Science & Medice, "Journal of Sports Science & Medice," *Reliability of the One-Repetition Maximum Test Based on Muscle Group and Gender,* pp. 221-225, 1 6 2012.

[29] P. A. Koen, "Fuzzy Front End: Effective Methods, Tools, and Techniques," [Online].

[30] D. Pereira, "O que é o Business Model Canvas," 8 7 2016. [Online]. Available: https://analistamodelosdenegocios.com.br/o-que-e-o-business-model-canvas/. [Accessed 11 2 2019].

[31] C. C. Aggarwal, Recommender Systems, NY, USA: Springer, 2016.

[32] F. Ricci, L. Rokach and B. Shapira, Introduction to Recommender SystemsHandbook, Springer, 2011.

[33] B. Schafer, D. Frankowski, J. Herlocker and S. Shilad , Collaborative Filtering Recommender Systems, Berlin: Springer, 2007.

[34] D. Jannach, M. Zanker, A. Felfernig and G. Friedrich, Recommender Systems: An Introduction, Cambridge, 2011.

[35] S. Luo, "Introduction to Recommender System," 10 12 2018. [Online]. Available: https://towardsdatascience.com/intro-to-recommender-system-collaborative-filtering-64a238194a26. [Accessed 12 9 2019].

[36] C. C. Aggarwal, "An Introduction to Recommender Systems," *Recommender Systems: The Textbook,* pp. 1-28, 2016.

[37] 王斌, "Comparison of User-Based and Item-Based Collaborative Filtering," 9 3 2018. [Online]. Available: https://medium.com/@wwwbbb8510/comparison-of-user-based-and-item-based-collaborative-filtering-f58a1c8a3f1d. [Accessed 19 9 2019].

[38] M. Singh, "Scalability and sparsity issues in recommender datasets: a survey," in *Knowledge and Information Systems*, 2018.

[39] G. Adomavicius , R. Sankaranarayanan , S. Sen and A. Tuzhilin , Incorporating Contextual Information in Recommender Systems Using a Multidimensional Approach.

[40] G. Adomavicius and A. Tuzhilin, "Toward the Next Generation of RecommenderSystems: A Survey of the State-of-the-Art andPossible Extensions," *IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERIN,* vol. 17, no. 6, pp. 734-749, 2005.

[41] P. Melville, R. J. Mooney and R. Nagarajan, "Content-Boosted Collaborative Filtering for Improved Recommendations," *Proceedings of the Eighteenth National Conference on Artificial Intelligence(AAAI-2002),* pp. 187-192, 7 2002.

[42] D. Billsus and M. J. Pazzani, Learning Collaborative Information Filters, Irvine.

[43] G. H. Golub and C. Reinsch, "Singular Value Decomposition and Least Squares Solutions," in *Linear Algebra*, pp. 134-151.

[44] J. Brownlee, "A Gentle Introduction fo Matrix Factorization for Machine Learning," 9 8 2019. [Online]. Available: https://machinelearningmastery.com/introduction-to-matrix-decompositions-for-machine-learning/. [Accessed 13 9 2019].

[45] N. Hug, "Understanding matrix factorization for recommendation (part 2) - the model behind SVD," 15 6 2017. [Online]. Available: http://nicolas-hug.com/blog/matrix_facto_2. [Accessed 13 9 2019].

[46] B. Schafer and D. Frankowski, Collaborative Filtering Recommender Systems, 2007.

[47] S. Xiaoyuan, A Survey of Collaborative Filtering Techniques, 2009.

[48] S. Deerwester, G. Furnas, S. Dumais, Landauer and Thomas, Indexing By Latent Semantic Analysis, 1990.

[49] M. Claypool, T. Mir, A. Gokhale and P. Murnikov, Combining Content-Based and Collaborative Filters in an Online Newspaper, 1999.

[50] S. Lam and J. Riedl, Shilling Recommender Systems for Fun and Profit, Minneapolis.

[51] B. Mobasher, R. Burke, R. Bhaumik and C. Williams, EffectiveAttackModelsforShillingItem-BasedCollaborativeFilteringSystems, Chicago.

[52] M. O'Mahony, N. Hurley, N. Kushmerick and G. Silvestre, Collaborative recommendation:A robustness analysis, Ireland.

[53] R. M. Bell and Y. Koren, Improved Neighborhood-based Collaborative Filtering.

[54] S. Khusro, Z. Ali and I. Ullah, Recommender Systems: Issues, Challenges, and Research Opportunities, Singapore: Springer, 2016.

[55] H. Polat and W. Du , Privacy-Preserving Collaborative Filtering, 2005.

[56] M. Heupel, M. Bourimi, L. Fischer and S. Scerri, Ontology-Enabled Access Control and Privacy Recommendations, Springer, 2014.

[57] C. Shahabi and Y.-S. Chen, Web Information Personalization: Challenges and Approaches, Los Angeles: Springer, 2003.

[58] Oxford dictionaries, "powerlifting," [Online]. Available: https://en.oxforddictionaries.com/definition/powerlifting. [Accessed 1 2 2019].

[59] Oxford dictionaries, "Olympic Weightlifting," [Online]. Available: https://en.oxforddictionaries.com/definition/olympic_weightlifting. [Accessed 1 2 2019].

[60] M. Emery, "Men's Bodybuilding: A Short History," 1 2003. [Online]. Available: http://www.bodybuildingreviews.net/Bodybuilding.html. [Accessed 1 2 2019].

[61] "Scientific Principles of Strength Training," 25 1 2018. [Online]. Available: https://www.jtsstrength.com/scientific-principles-strength-training/. [Accessed 6 9 2019].

[62] "Arnold Schwarzenegger's 8 Best Training Principles," 7 8 2018. [Online]. Available: https://www.bodybuilding.com/content/arnold-schwarzenegger-8-best-training-principles.html. [Accessed 6 9 2019].

[63] Oxford dictionaries, "Athletic," [Online]. Available: https://en.oxforddictionaries.com/definition/athletic. [Accessed 1 2 2019].

[64] AmazinGym, "AmazinGym," [Online]. Available: https://amazingym.pt/. [Accessed 1 2 2019].

[65] AmazinGym, "AmazinGym," [Online]. Available: https://play.google.com/store/apps/details?id=digifit.android.virtuagym.pro.amazingymmatosinhos&hl=en_US. [Accessed 1 2 2019].

[66] AmazinGym, "AmazinGym," [Online]. Available: https://itunes.apple.com/us/app/amazingym/id1283090572?mt=8. [Accessed 1 2 2019].

[67] Tripla Forma, "Tripla Forma," [Online]. Available: http://www.triplaforma.com/. [Accessed 1 2 2019].

[68] Tripla Forma, "Tripla Forma," [Online]. Available: https://play.google.com/store/apps/details?id=com.onvirtualgym.tripla_Forma&hl=en_US. [Accessed 1 2 2019].

[69] tanita, "How BIA works," [Online]. Available: https://www.tanita.com/en/howbiaworks/. [Accessed 1 2 2019].

[70] Intensity, "Intensity," 2019. [Online]. Available: https://www.intensityapp.com/. [Accessed 1 2 2019].

[71] Intensity, "Intensity - Powerlifting Workout Tracker & Gym Log," [Online]. Available: https://play.google.com/store/apps/details?id=com.taylorhamling.intensity&hl=en_US. [Accessed 1 2 2019].

[72] Intensity, "Intensity - Workout Journal," [Online]. Available: https://itunes.apple.com/us/app/intensity-workout-journal/id1047407323. [Accessed 1 2 2019].

[73] Strong, "Strong," 2018. [Online]. Available: https://www.strong.app/. [Accessed 1 2 2019].

[74] Strong, "Strong: Exercise Gym Log, 5x5," [Online]. Available: https://play.google.com/store/apps/details?id=io.strongapp.strong&referrer=utm_source%3Dwebsite. [Accessed 1 2 2019].

[75] Strong, "Strong Workout Tracker Gym Log," [Online]. Available: https://itunes.apple.com/app/apple-store/id464254577?mt=8. [Accessed 1 2 2019].

[76] Nike, "Nike Training Club App," [Online]. Available: https://www.nike.com/us/en_us/c/nike-plus/training-app. [Accessed 1 2 2019].

[77] Nike, "Nike Training Club - Workouts & Fitness Guidance," [Online]. Available: https://play.google.com/store/apps/details?id=com.nike.ntc&hl=en_US. [Accessed 1 2 2019].

[78] Nike, "Nike Training Club," [Online]. Available: https://itunes.apple.com/app/nike+-training-club-workouts/id301521403. [Accessed 1 2 2019].

[79] Freeletics, "Freeletics," [Online]. Available: https://www.freeletics.com/en/. [Accessed 1 2 2019].

[80] Freeletics, "Freeletics: Personal Fitness Coach & Body Workouts," [Online]. Available: https://play.google.com/store/apps/details?id=com.freeletics.lite&hl=en_US. [Accessed 1 2 2019].

[81] Freeletics, " Freeletics: Workout & Fitness," [Online]. Available: https://itunes.apple.com/app/freeletics-functional-high/id654810212?ls=1&mt=8. [Accessed 1 2 2019].

[82] Freeletics, "Freeletics Coach Explained," [Online]. Available: https://help.freeletics.com/hc/en-us/articles/360001805259-Freeletics-Coach-Explained. [Accessed 1 2 2019].

[83] Jefit, "Jefit," [Online]. Available: https://www.jefit.com/. [Accessed 1 2 2019].

[84] Jefit, "JEFIT Workout Tracker, Weight Lifting, Gym Log App," [Online]. Available: https://play.google.com/store/apps/details?id=je.fit&hl=en_US. [Accessed 1 2 2019].

[85] Jefit, "JEFIT Workout Planner Gym Log," [Online]. Available: https://itunes.apple.com/app/apple-store/id449810000?mt=8. [Accessed 1 2 2019].

[86] Jefit, "Jefit Elite," [Online]. Available: https://www.jefit.com/elite/. [Accessed 1 2 2019].

[87] Fitbod, "Fitbod," [Online]. Available: https://www.fitbod.me/. [Accessed 1 2 2019].

[88] Business Dictionary, "stakeholder," [Online]. Available: http://www.businessdictionary.com/definition/stakeholder.html. [Accessed 13 2 2019].

[89] Agile Modeling, "UML 2 Use Case Diagrams: An Agile Introduction," [Online]. Available: http://agilemodeling.com/artifacts/useCaseDiagram.htm. [Accessed 12 2 2019].

[90] U. Eriksson, "Functional vs Non Functional Requirements," 5 4 2012. [Online]. Available: https://reqtest.com/requirements-blog/functional-vs-non-functional-requirements/. [Accessed 13 2 2019].

[91] IBM, "What, no supplementary specification?," 1 7 2004. [Online]. Available: https://www.ibm.com/developerworks/rational/library/3975.html. [Accessed 13 2 2019].

[92] P. Brown, "What is the Domain Model in Domain Driven Design?," 12 9 2014. [Online]. Available: https://culttt.com/2014/11/12/domain-model-domain-driven-design/. [Accessed 10 9 2019].

[93] ISEP, "Documentação Arquitetural," 2016. [Online]. [Accessed 13 2 2019].

[94] ISEP, "Software Architecture Document," 2016. [Online]. [Accessed 13 2 2019].

[95] Technopedia, "Back Office Application," [Online]. Available: https://www.techopedia.com/definition/1406/back-office-application. [Accessed 13 2 2019].

[96] Techopedia, "Front Office Application," [Online]. Available: https://www.techopedia.com/definition/28460/front-office-application. [Accessed 13 2 2019].

[97] BBVA, "8 advantages of APIs for developers," [Online]. Available: https://bbvaopen4u.com/en/actualidad/8-advantages-apis-developers. [Accessed 13 2 2019].

[98] C. Richardson, "Pattern: Microservice Architecture," [Online]. Available: https://microservices.io/patterns/microservices.html. [Accessed 13 2 2019].

[99] Tutorials Point, "UML - Deployment Diagrams," [Online]. Available: https://www.tutorialspoint.com/uml/uml_deployment_diagram.htm. [Accessed 12 9 2019].

[100] P. Kruchten, The “4+1” View Model of Software Architecture, 1995.

[101] M. Rouse, "intelligent system," [Online]. Available: https://whatis.techtarget.com/definition/intelligent-system. [Accessed 19 9 2019].

[102] S. Clayton, "Building a Recommedantion Engine in C#," 22 3 2018. [Online]. Available: https://www.codeproject.com/Articles/1232150/Building-a-Recommendation-Engine-in-Csharp. [Accessed 23 9 2019].

[103] F. M. Harper and J. A. Konstan, "The MovieLens Datasets: History and Context," p. 20, 2015.

[104] A. Williams, "How to cross-validate PCA, clustering, and matrix decomposition models," 26 2 2018. [Online]. Available: http://alexhwilliams.info/itsneuronalblog/2018/02/26/crossval/. [Accessed 22 9 2019].

[105] S. M, "Why and how to Cross Validate a Model?," 13 9 2018. [Online]. Available: https://towardsdatascience.com/why-and-how-to-cross-validate-a-model-d6424b45261f. [Accessed 22 9 2019].

[106] M. D. Ekstrand and J. A. Konstan, "Matrix Factorization and Advanced Techniques," [Online]. Available: https://www.coursera.org/learn/matrix-factorization. [Accessed 22 9 2019].

[107] P. Pandey, "Understanding the Mathematics behind Gradient Descent.," 18 3 2019. [Online]. Available: https://towardsdatascience.com/understanding-the-mathematics-behind-gradient-descent-dde5dc9be06e. [Accessed 21 9 2019].

[108] W. Kirwin, "Implicit Recommender Systems: Biased Matrix Factorization¶," 11 1 2016. [Online]. Available: http://activisiongamescience.github.io/2016/01/11/Implicit-Recommender-Systems-Biased-Matrix-Factorization/. [Accessed 21 9 2019].

[109] A. A. Yeung, "Matrix Factorization: A Simple Tutorial and Implementation in Python," 23 4 2017. [Online]. Available: http://www.albertauyeung.com/post/python-matrix-factorization/. [Accessed 21 9 2019].

[110] C. McDonald, "Machine learning fundamentals (I): Cost functions and gradient descent," 27 9 2017. [Online]. Available: https://towardsdatascience.com/machine-learning-fundamentals-via-linear-regression-41a5d11f5220. [Accessed 21 9 2019].

[111] T. Roughgarden and G. Valiant, CS168: The Modern Algorithmic ToolboxLecture #6: Stochastic Gradient Descent andRegularization, 2016.

[112] I. Dabbura, "Gradient Descent Algorithm and Its Variants," 21 12 2017. [Online]. Available: https://towardsdatascience.com/gradient-descent-algorithm-and-its-variants-10f652806a3. [Accessed 22 9 2019].

[113] A. Al-Masri, "What Are Overfitting and Underfitting in Machine Learning?," 21 6 2019. [Online]. Available: https://towardsdatascience.com/what-are-overfitting-and-underfitting-in-machine-learning-a96b30864690. [Accessed 22 9 2019].

[114] V. Chekka, "Regularization in Machine Learning: Connect the dots," 30 8 2018. [Online]. Available: https://towardsdatascience.com/regularization-in-machine-learning-connecting-the-dots-c6e030bfaddd. [Accessed 22 9 2019].

[115] Kent State University Libraries, "PSS Tutorials: Pearson Correlation," 12 8 2019. [Online]. Available: https://libguides.library.kent.edu/SPSS/PearsonCorr. [Accessed 22 9 2019].

[116] A. Gunawardana and G. Shani, "A Survey of Accuracy Evaluation Metrics of Recommendation Tasks," *Journal of Machine Learning Research,* no. 10, pp. 2935-2962, 2009.

[117] A. Agarwal and M. Chauhan, "Similarity Measuresused in Recommender Systems: A Study," *International Journal of EngineeringTechnology Science and Research,* vol. 4, no. 6, p. 2394–3386, 2017.

[118] J. Wesner, "MAE and RMSE — Which Metric is Better?," 23 3 2016. [Online]. Available: https://medium.com/human-in-a-machine-world/mae-and-rmse-which-metric-is-better-e60ac3bde13d. [Accessed 23 9 2019].

[119] A. Bari, M. Chaouchi and T. Jung, Predictive Analytics For Dummies, 2014.

[120] B. Sarwar, G. Karypis, G. Cybenko and J. Tsibouklis, "Item-based collaborative filtering recommendation algorithms," 1 2001.

[121] R. Burke, "Hybrid Recommender Systems: Survey and Experiments," *User Modeling and User-Adapted Interaction,* 11 2002.

[122] R. Sunil, "7 Regression Techniques you should know!," 14 8 2015. [Online]. Available: https://www.analyticsvidhya.com/blog/2015/08/comprehensive-guide-regression/. [Accessed 16 9 2019].

[123] E. Gamma, Padrões de Projeto: Soluções reutilizáveis de software orientado a objetos., Bookman, 2000.

[124] Purple Math, "Straight-Line Equations: Slope-Intercept Form," [Online]. Available: https://www.purplemath.com/modules/strtlneq.htm. [Accessed 17 9 2019].

[125] B. Eisenman, "Chapter 1. What Is React Native?," [Online]. Available: https://www.oreilly.com/library/view/learning-react-native/9781491929049/ch01.html. [Accessed 24 9 2019].

[126] M. Newton, "Understanding Expo for React Native," [Online]. Available: https://hackernoon.com/understanding-expo-for-react-native-7bf23054bbcd. [Accessed 24 9 2019].

[127] Visual Studio Code, "Getting Started," [Online]. Available: https://code.visualstudio.com/docs. [Accessed 24 9 2019].

[128] Redux, "Getting Started with Redux," [Online]. Available: https://redux.js.org/introduction/getting-started. [Accessed 24 9 2019].

[129] Log Rocket, "Why use Redux? Reasons with clear examples," 31 8 2018. [Online]. Available: https://blog.logrocket.com/why-use-redux-reasons-with-clear-examples-d21bffd5835/. [Accessed 24 9 2019].

[130] Redux, "Actions," [Online]. Available: https://redux.js.org/basics/actions. [Accessed 24 9 2019].

[131] Redux, "Store," [Online]. Available: https://redux.js.org/api/store. [Accessed 24 9 2019].

[132] Redux, "Reducers," [Online]. Available: https://redux.js.org/basics/reducers. [Accessed 24 9 2019].

[133] React Redux, "connect()," [Online]. Available: https://react-redux.js.org/api/connect. [Accessed 26 9 2019].

[134] React Native, "AsyncStorage," [Online]. Available: https://facebook.github.io/react-native/docs/asyncstorage. [Accessed 24 9 2019].

[135] React Native, "Navigating Between Screens," [Online]. Available: https://facebook.github.io/react-native/docs/navigation. [Accessed 25 9 2019].

[136] React Native, "Tab navigation," [Online]. Available: https://reactnavigation.org/docs/en/tab-based-navigation.html. [Accessed 25 9 2019].

[137] React Navigation, "Drawer navigation," [Online]. Available: https://reactnavigation.org/docs/en/drawer-based-navigation.html. [Accessed 25 9 2019].

[138] React Native, "createStackNavigator," [Online]. Available: https://reactnavigation.org/docs/en/stack-navigator.html. [Accessed 25 9 201].

[139] React Native, "createSwitchNavigator," [Online]. Available: https://reactnavigation.org/docs/en/switch-navigator.html. [Accessed 25 9 2019].

[140] WhatIs, "internationalization (I18N)," [Online]. Available: https://whatis.techtarget.com/definition/internationalization-I18N. [Accessed 24 9 2019].

[141] J. Arvidsson, "react-native-animatable," [Online]. Available: https://github.com/oblador/react-native-animatable. [Accessed 25 9 2019].

[142] J. Lauritzen, "react-native-app-intro-slider," [Online]. Available: https://github.com/Jacse/react-native-app-intro-slider. [Accessed 25 9 2019].

[143] T. Mecinskas, "react-native-calendars," [Online]. Available: https://github.com/wix/react-native-calendars. [Accessed 25 9 2019].

[144] J. Arvidsson, "react-native-collapsible," [Online]. Available: https://github.com/oblador/react-native-collapsible. [Accessed 25 9 2019].

[145] M. Mazzarolo, "react-native-dialog," [Online]. Available: https://github.com/mmazzarolo/react-native-dialog. [Accessed 25 9 2019].

[146] React Native Elements , "react-native-elements," [Online]. Available: https://github.com/react-native-training/react-native-elements. [Accessed 25 9 2019].

[147] F. Safi, "react-native-gifted-chat," [Online]. Available: https://github.com/FaridSafi/react-native-gifted-chat. [Accessed 25 9 2019].

[148] N. Baugh, "react-native-loading-spinner-overlay," [Online]. Available: https://github.com/joinspontaneous/react-native-loading-spinner-overlay. [Accessed 25 9 2019].

[149] M. Milyutin, "react-native-material-menu," [Online]. Available: https://github.com/mxck/react-native-material-menu. [Accessed 25 9 2019].

[150] A. Nazarov, "react-native-material-textfield," [Online]. Available: https://github.com/n4kz/react-native-material-textfield. [Accessed 25 9 2019].

[151] M. Mazzarolo, "react-native-modal," [Online]. Available: https://github.com/react-native-community/react-native-modal. [Accessed 25 9 2019].

[152] D. Urbaniak, "react-native-paper," [Online]. Available: https://github.com/callstack/react-native-paper. [Accessed 25 9 2019].

[153] A. Vitanov, "react-native-parallax-scroll-view," [Online]. Available: https://github.com/i6mi6/react-native-parallax-scroll-view. [Accessed 25 9 2019].

[154] zooble, "react-native-picker," [Online]. Available: https://github.com/beefe/react-native-picker. [Accessed 25 9 2019].

[155] LawnStarter, "react-native-picker," [Online]. Available: https://github.com/lawnstarter/react-native-picker-select. [Accessed 25 9 2019].

[156] J. Arvidsson, "react-native-progress," [Online]. Available: https://github.com/oblador/react-native-progress. [Accessed 25 9 2019].

[157] H. lee, "react-native-pure-chart," [Online]. Available: https://github.com/oksktank/react-native-pure-chart. [Accessed 25 9 2019].

[158] R. Caferati, "react-native-really-awesome-button," [Online]. Available: https://github.com/rcaferati/react-native-really-awesome-button. [Accessed 25 9 2019].

[159] Agiletech, "react-native-search-box," [Online]. Available: https://github.com/react-native-vietnam/react-native-search-box. [Accessed 25 9 2019].

[160] Ren, "react-native-sectioned-multi-select," [Online]. Available: https://github.com/renrizzolo/react-native-sectioned-multi-select. [Accessed 25 9 2019].

[161] B. Delmaire, "react-native-snap-carousel," [Online]. Available: https://github.com/archriss/react-native-snap-carousel. [Accessed 25 9 2019].

[162] C. Wood, "react-native-sortable-listview," [Online]. Available: https://github.com/deanmcpherson/react-native-sortable-listview. [Accessed 25 9 2019].

[163] M. Stevens, "react-native-stopwatch-timer," [Online]. Available: https://github.com/michaeljstevens/react-native-stopwatch-timer. [Accessed 25 9 2019].

[164] J. Hanson, "react-native-swipeable," [Online]. Available: https://github.com/jshanson7/react-native-swipeable. [Accessed 25 9 2019].

[165] Microsoft, "Introduction to ASP.NET Identity," 22 1 2019. [Online]. Available: https://docs.microsoft.com/en-us/aspnet/identity/overview/getting-started/introduction-to-aspnet-identity. [Accessed 30 9 2019].

[166] Microsoft, " Rijndael Class," [Online]. Available: https://docs.microsoft.com/en-us/dotnet/api/system.security.cryptography.rijndael?view=netframework-4.8.

[167] TechTarget, "Advanced Encryption Standard (AES)," [Online]. Available: https://searchsecurity.techtarget.com/definition/Advanced-Encryption-Standard. [Accessed 29 9 2019].

[168] Microsoft, " Rfc2898DeriveBytes Class," [Online]. Available: https://docs.microsoft.com/en-

us/dotnet/api/system.security.cryptography.rfc2898derivebytes?view=netframework-4.8.

[169] Microsoft, " System.Security.Cryptography Namespace," [Online]. Available: https://docs.microsoft.com/en-us/dotnet/api/system.security.cryptography?view=netframework-4.8. [Accessed 29 9 2019].

[170] S. Verma, S. K. Pal and S. K. Muttoo, "A new tool for lightweight encryption on android," IEEE, Gurgaon, India, 2014.

[171] Microsoft, "Code First Migrations," 23 10 2016. [Online]. Available: https://docs.microsoft.com/en-us/ef/ef6/modeling/code-first/migrations/. [Accessed 30 9 2019].

[172] Microsoft, "Migrations," 10 5 2018. [Online]. Available: https://docs.microsoft.com/en-us/ef/core/managing-schemas/migrations/. [Accessed 30 9 2019].

[173] Microsoft, "Language Understanding (LUIS)," [Online]. Available: https://www.luis.ai/home. [Accessed 1 10 2019].

[174] Microsoft, "What is Language Understanding (LUIS)?," 11 6 2019. [Online]. Available: https://docs.microsoft.com/en-us/azure/cognitive-services/luis/what-is-luis. [Accessed 1 10 2019].

[175] Microsoft, "Concepts about intents in your LUIS app," 29 7 2019. [Online]. Available: https://docs.microsoft.com/en-us/azure/cognitive-services/luis/luis-concept-intent. [Accessed 1 10 2019].

[176] Microsoft, "Entity types and their purposes in LUIS," 24 7 2019. [Online]. Available: https://docs.microsoft.com/en-us/azure/cognitive-services/luis/luis-concept-entity-types. [Accessed 1 10 2019].

[177] Microsoft, "Understand what good utterances are for your LUIS app," 5 7 2019. [Online]. Available: https://docs.microsoft.com/en-us/azure/cognitive-services/luis/luis-concept-utterance. [Accessed 1 10 2019].

[178] Microsoft, "Key concepts in Direct Line API 3.0," 6 1 2019. [Online]. Available: https://docs.microsoft.com/en-us/azure/bot-service/rest-api/bot-framework-rest-direct-line-3-0-concepts?view=azure-bot-service-4.0. [Accessed 1 10 2019].

[179] Central Intelligence Agency, The World Factbook, 2006.

[180] Stanford, "Evaluation Metrics," [Online]. Available: http://cs229.stanford.edu/section/evaluation_metrics.pdf. [Accessed 16 2 2019].

[181] Medium, "Mean Absolute Error ~ MAE [Machine Learning(ML)]," [Online]. Available: https://medium.com/@ewuramaminka/mean-absolute-error-mae-machine-learning-ml-b9b4afc63077. [Accessed 4 10 2019].

[182] Microsoft, " Stopwatch Class," [Online]. Available: https://docs.microsoft.com/en-us/dotnet/api/system.diagnostics.stopwatch?view=netframework-4.8. [Accessed 5 10 2019].

[183] T. Chai and R. R. Draxler, "Root mean square error (RMSE) or mean absolute error (MAE)? –Arguments against avoiding RMSE in the literature," 2014.

[184] T. Rackaitis, "Evaluating Recommender Systems: Root Means Squared Error or Mean Absolute Error?," [Online]. Available: https://towardsdatascience.com/evaluating-recommender-systems-root-means-squared-error-or-mean-absolute-error-1744abc2beac. [Accessed 7 10 2019].

[185] G. E. Forsythe and C. B. Moler, Computer solution of linear algebraic systems, Englewood Cliffs, New Jersey: Prentice-Hall, 1967.

[186] G. Drakos, "How to select the Right Evaluation Metric for Machine Learning Models: Part 1 Regression Metrics," 26 8 2018. [Online]. Available: https://towardsdatascience.com/how-to-select-the-right-evaluation-metric-for-machine-learning-models-part-1-regrression-metrics-3606e25beae0. [Accessed 22 9 2019].

Appendix A

Análise de Utilização de Aplicações de Ginásio/Fitness

Este questionário é realizado no âmbito da disciplina Tese de Mestrado (TMDEI) no curso Mestrado de Engenharia de Software no Instituto Superior de Engenharia do Porto.
O objetivo é recolher dados acerca da utilização de aplicações fitness, bem como da sua não utilização.
Agradecemos, desde já, a sua disponibilidade ao preencher o questionário, realçando, que, todos os dados inseridos são confidenciais e que serão apenas usados para análise deste projeto.
Prevemos que o seu preenchimento seja breve, não demorando mais do que 3 minutos.
Qualquer dúvida ou sugestão por favor contactar: 1140400@isep.ipp.pt ou 1140402@isep.ipp.pt

*Obrigatório

Objetivos

Tendo em conta que o tema da Tese de Mestrado é o desenvolvimento de uma aplicação fitness, este questionário tem como objetivo analisar vários aspetos da utilização de aplicações relacionados com o presente tema, tais como:

- objetivos da sua utilização;
- quais as funcionalidades mais importantes;
- principais razões de como as aplicações fitness podem ajudar a atingir os objetivos específicos;
- funcionalidades que poderiam ser inseridas;
- analisar o porquê da sua não utilização;

Dados pessoais

1. **Sexo** *
   Marcar apenas uma oval.
   - ◯ Masculino
   - ◯ Feminino

2. **Idade** *

3. **Frequência de utilização do ginásio** *
   Marcar apenas uma oval.
   - ◯ 1-2x por semana
   - ◯ 2-3x por semana
   - ◯ 3-4x por semana
   - ◯ 4-5x por semana
   - ◯ 5x por semana ou mais

4. **Há quanto tempo anda no ginásio?** *

Marcar apenas uma oval.

- ◯ Menos de 6 meses
- ◯ 6 meses a 2 anos
- ◯ 2 a 3 anos
- ◯ 3 a 5 anos
- ◯ 5 a 10 anos
- ◯ Mais de 10 anos

5. **Objetivos no ginásio (selecione 1 ou mais)** *

Marcar tudo o que for aplicável.

- ☐ Ganhar músculo
- ☐ Perder peso
- ☐ Ganhar força
- ☐ Preparação física (orientada a algum desporto/atividade física)
- ☐ Ganhar peso
- ☐ Manter estilo de vida saudável

Plano de treino

6. **Segue um plano de treino?** *

Marcar apenas uma oval.

- ◯ Não
- ◯ Sim, feito por mim
- ◯ Sim, feito pelo Personal Trainer
- ◯ Sim, feito por uma aplicação fitness

7. **Regista plano de treino?** *

Marcar apenas uma oval.

- ◯ Não
- ◯ Sim, em papel
- ◯ Sim, no Excel ou semelhante
- ◯ Sim, numa aplicação fitness

Uso de aplicações fitness

8. Utiliza ou já utilizou aplicações fitness (para registo de dados, planeamento, etc)? *

Marcar apenas uma oval.

- ◯ Utilizo neste momento *Passe para a pergunta 10.*
- ◯ Já utilizei, mas não utilizo neste momento *Passe para a pergunta 14.*
- ◯ Não utilizo e nunca utilizei *Passe para a pergunta 9.*

Não utiliza aplicações de fitness

9. Porque nunca utilizou aplicações fitness? *

Marcar apenas uma oval.

- ◯ Não tenho interesse *Pare de preencher este formulário.*
- ◯ Não uso dispositivos eletrónicos durante o treino *Pare de preencher este formulário.*
- ◯ Ainda não encontrei uma que tivesse as funcionalidades desejadas *Passe para a pergunta 16.*
- ◯ Tenho um treino demasiado específico *Passe para a pergunta 16.*
- ◯ Outra: _____ *Pare de preencher este formulário.*

Pare de preencher este formulário.

Utilização

10. Qual o nome da aplicação fitness que usa?

_____

11. Há quanto tempo usa aplicações fitness?

Marcar apenas uma oval.

- ◯ 2 semanas ou menos
- ◯ Entre 2 a 4 semanas
- ◯ Entre 1 a 3 meses
- ◯ Entre 3 a 12 meses
- ◯ Mais de 1 ano

12. Quais as razões pelas quais usa a aplicação fitness? (selecione 1 ou mais) *

Marcar tudo o que for aplicável.

- [ ] Aplicação usada pelo ginásio que frequento
- [ ] Marcação de aulas
- [ ] Visualizar informações de exercícios e planos
- [ ] Consultar progresso (peso, % gordura, etc)
- [ ] Registar treinos realizados
- [ ] Planear treinos
- [ ] Acompanhar o treino em tempo real
- [ ] Comunicar com Personal Trainer
- [ ] Ver planos de nutrição
- [ ] Publicar nas redes sociais
- [ ] Sugestão automática de exercícios/treinos/planos por parte da aplicação
- [ ] Outra: _____

13. Pontos negativos da aplicação fitness que usa (selecione 1 ou mais) *

Marcar tudo o que for aplicável.

- [ ] Publicidade exagerada
- [ ] Registo de dados exaustivo
- [ ] Preço elevado
- [ ] Experiência de utilização medíocore
- [ ] Falta de funcionalidades
- [ ] Notificações incómodas
- [ ] Não existem pontos negativos
- [ ] Outra: _____

Passe para a pergunta 15.

Não Utilização

14. Porque deixou de usar aplicações fitness ? (selecione 1 ou mais) *

Marcar tudo o que for aplicável.

- [ ] Publicidade exagerada
- [ ] Registo de dados exaustivo
- [ ] Preço elevado
- [ ] Experiência de utilização pouco intuitiva
- [ ] Falta de funcionalidades
- [ ] Notificações incómodas
- [ ] Outra: _____

Passe para a pergunta 15.

Dispositivo de utilização

15. Em que dispositivo utiliza maioritariamente as aplicações de fitness? *

Marcar apenas uma oval.

- ◯ Android
- ◯ Apple
- ◯ Web Browser
- ◯ Outra: _____

Nova aplicação fitness

16. Gostaria que existisse uma aplicação de fitness que facilitasse o alcance dos seus objetivos? *

Marcar apenas uma oval.

- ◯ Sim
- ◯ Não *Pare de preencher este formulário.*

Novas funcionalidades

Selecionar, em baixo, as funcionalidades que acharia mais atrativas numa nova aplicação fitness. Caso tenha alguma ideia não listada, adicionar em "Outra".

17. Que novas funcionalidades gostaria de ver numa nova aplicação fitness? (selecione 1 ou mais)

Marcar tudo o que for aplicável.

- ☐ Recomendação automática de exercícios/treinos/planos por parte da aplicação
- ☐ Acompanhamento por um Coach/PT (Monitorização da evolução, sugestões, ajuda, etc.)
- ☐ Assistente Pessoal Virtual (Esclarecimento de dúvidas de fitness com um "robô")
- ☐ Sincronização com SmartWatches
- ☐ Visualizar artigos científicos sobre Fitness
- ☐ Outra: _____

Appendix B

Inquérito à aplicação fitness

1. **Aplicação Móvel**

Marcar apenas uma oval por linha.

	Discordo completamente	Discordo	Sem opinião	Concordo	Concordo completamente
A aplicação possui uma boa apresentação/design.					
A utilização da aplicação é intuitiva e simples.					
A aplicação é rápida.					
A aplicação está adaptada ao meu dispositivo.					
O processo de registo e login é realizado de forma rápida.					
O processo de criação de treinos e a sua realização é feita de forma simples e eficaz.					
É possível criar planos e programas de treino com todos os pormenores necessários à sua realização.					
Os filtros apresentados para os exercícios, treinos, entre outros, são eficientes na procura do desejado.					
A aplicação é de fácil configuração.					
A aplicação poderia ser útil enquanto pratico exercício físico (ginásio).					
Recomendaria a aplicação a qualquer praticante de ginásio.					
Classificaria a aplicação como muito boa.					

2. **Assistente Pessoal**

Marcar apenas uma oval por linha.

	Discordo completamente	Discordo	Sem opinião	Concordo	Concordo completamente
	Discordo completamente	Discordo	Sem opinião	Concordo	Concordo completamente
A conversa com o assistente pessoal é fluída.					
O assistente pessoal consegue interpretar e responder com sentido ao que foi dito.					
O assistente pessoal esclarece com qualidade as dúvidas sobre os exercícios.					
O assistente pessoal é uma mais valia para a aplicação.					

3. **Sistema de Recomendação Condicional**

Marcar apenas uma oval por linha.

	Discordo completamente	Discordo	Sem opinião	Concordo	Concordo completamente
As recomendações sugeridas são boas.					
Os treinos recomendados fazem sentido tendo em conta os meus gostos/perfil.					

Appendix C

Demonstração da Recomendação Condicional com Dados Reais

Em primeiro lugar, os dados dos treinos passados do especialista, juntamente com os seus dados preferenciais (p.e. lista de objetivos, tipos de treino, etc.) e a lista de treinos para recomendação, existentes na base de dados foram fornecidos ao algoritmo.

Devido a restrições de tempo, a base de dados de treinos para recomendação é relativamente reduzida, mas com diversidade suficiente para permitir a demonstração da aplicação de todos os filtros. Na tabela seguinte pode-se visualizar a informação destes mesmos, juntamente com as características que os definem.

Tabela 1 -Treinos na base de dados

ID	Nome	Nível	Objetivos	Tipos de Treino	Músculos	Área do Corpo	Intensidade [0-3]	Volume (Reps)
1	Chest	Intermédio	Ganhar Músculo, Ganhar Força	Hipertrofia, Força	Braços, Peito	Upper	1.6	150
2	Legs	Intermédio	Ganhar Músculo, Ganhar Força	Hipertrofia, Força	Glúteos, Upper Leg, Lower Leg	Lower	2	164
3	MAX Lower Body	Intermédio	Ganhar Músculo, Ganhar Força	Hipertrofia, Força	Glúteos, Upper Leg, Lower Leg	Lower	2.25	128
4	High Level Weightlifting	Avançado	Ganhar Músculo, Ganhar Força	Força, Weightlifting	Glúteos, Upper Leg, Upper Back, Core	Full	2.5	60
5	Low Level Weightlifting - Competition Peaking	Iniciante	Competição	Força, Weightlifting	Glúteos, Upper Leg, Upper Back, Core	Full	2.1	51
6	Low Level Weightlifting	Iniciante	Ganhar Músculo, Ganhar Força	Weightlifting	Glúteos, Upper Leg, Upper	Full	1.9	55

					Back, Core			
7	Upper Body Hypertrophy	Iniciante	Ganhar Músculo, Ganhar Força	Hipertrofia	Braços, Peito	Upper	1.5	166
8	Hypertrophy w/ Flexibility	Intermédio	Ganhar Músculo, Ganhar Força	Flexibilidade, Hipertrofia	Upper Leg, Lower Leg	Lower	1.8	137

Os dados do especialista – preferências e histórico de treinos – podem ser sumarizados nas seguintes tabelas, respetivamente.

Tabela 2 - Preferências do especialista

Nível	Objetivos	Tipos de Treino	Áreas do Corpo	Músculos
Experiente	Ganhar músculo	Hipertrofia, Powerlifting, Força, Flexibilidade	Full	Todos

Tabela 3 - Histórico de treinos do especialista

ID	Nome	Nível	Objetivos	Tipos de Treino	Músculos	Área do Corpo	Intensidade	Volume
1	Dia 1	Intermédio	Ganhar Músculo	Hipertrofia, Powerlifting, Força	Upper Back, Peito, Core, Braços, Ombros	Upper	2.7	133
2	Dia 2	Intermédio	Ganhar Músculo	Hipertrofia, Powerlifting, Força	Upper Back, Core, Braços, Ombros, Upper Leg	Lower	2.3	132
3	Dia 3	Intermédio	Ganhar Músculo	Hipertrofia, Powerlifting, Força	Upper Back, Core, Ombros	Upper	1.6	125
4	Dia 4	Intermédio	Ganhar Músculo	Hipertrofia, Powerlifting, Flexibilidade	Upper Back, Core, Ombros	Upper	1.6	171
5	Dia 5	Intermédio	Ganhar Músculo	Hipertrofia, Powerlifting, Força	Upper Back, Peito, Braços, Ombros	Upper	2.6	72
6	Dia 6	Intermédio	Ganhar Músculo	Hipertrofia, Powerlifting, Força	Upper Leg, Braços	Lower	2.5	165

Com o historial do utilizador em conta, foram calculadas as preferências em falta. Isto é, foram calculadas quais as preferências definidas que não foram encontradas no historial. Por exemplo, se os tipos de treino definidos sejam Hipertrofia, Força e Flexibilidade, e no historial existam treinos que

apresentem Hipertrofia e Flexibilidade, mas nenhum apresente Força, então Força será o tipo de treino desejado para a recomendação do novo treino. Do mesmo modo, caso todos os tipos de treino se encontrem no historial, então as preferências não são ajustadas. Com isso em conta, as novas preferências são as seguintes.

Tabela 4 – Preferências ajustadas

Nível	Objetivos	Tipos de Treino	Áreas do Corpo	Músculos
Experiente	Ganhar músculo	Hipertrofia, Powerlifting, Força, Flexibilidade	Full	Lower Leg, Glutes, Lower Back, Forearms

De seguida, os filtros das preferências ajustadas do especialista foram aplicados, resultando na remoção de alguns treinos da lista inicial, conforme a seguinte tabela.

Tabela 5 – Explicação da filtragem

ID	Explicação
1	Removido por não conter músculos usados que se alinhem com os do especialista.
2	Mantido.
3	Mantido.
4	Removido por não conter tipos de treino que se alinhem com os do especialista.
5	Removido por não conter objetivos que se alinhem com os do especialista.
6	Removido por não conter tipos de treino que se alinhem com os do especialista.
7	Removido por não conter músculos usados que se alinhem com os do especialista.
8	Mantido.

Tendo a lista reduzida apenas aos treinos que se identificam melhor com as preferências do especialista, é possível fazer desde já uma recomendação com os 3 treinos. De qualquer das formas, falta decidir qual o treino que melhor se adapta, comparando o volume e intensidade. Para tal, foi previsto um valor ideal para volume e intensidade com base nos treinos passados do especialista, como evidenciado nas figuras seguintes.

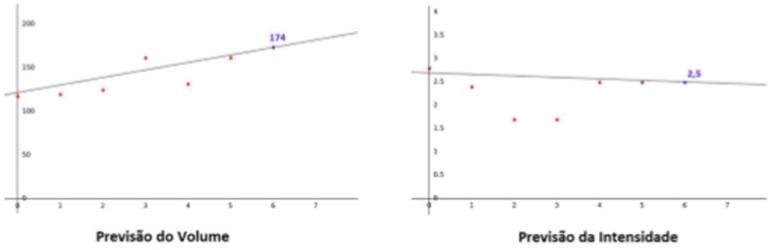

Figura 1 – Previsões para Volume e Intensidade

Cada ponto vermelho indica um valor de volume/intensidade nos treinos passados do especialista. Traçando uma *Line of Best Fit*, é possível prever que o próximo treino teria 174 de volume e 2.5 de intensidade. É possível também observar que há um evidente aumento de volume e uma ligeira descida de intensidade.

Com estes dados, é possível identificar qual dos 3 treinos se identifica melhor com os valores previstos. Para tal, o valor de volume e intensidade foi multiplicado, obtendo uma previsão de $174 \times 2.5 = 435$. Na seguinte tabela apresentam-se os valores desta multiplicação para os 3 treinos finalistas.

Tabela 6 – Valores de volume/intensidade das recomendações

ID	$\boldsymbol{volume \times intensidade}$	Diferença
2	$164 \times 2 = 328$	107
3	$128 \times 2.25 = 288$	147
8	$137 \times 1.8 = 246$	189

Com os dados da tabela anterior é possível verificar que o treino que melhor se adequa ao valor previsto da multiplicação do volume pela intensidade é o de ID 2, pois possui a menor diferença. Como tal, essa é a recomendação final.

Appendix D

Inquérito ao sistema de recomendação

1. **Sistema de Recomendação**
   Marcar apenas uma oval por linha.

	Discordo completamente	Discordo	Sem opinião	Concordo	Concordo completamente
As recomendações sugeridas são boas.					
Os treinos recomendados fazem sentido tendo em conta os meus gostos/perfil.					
A avaliação do volume e intensidade é eficiente.					
A forma de previsão do volume e intensidade é eficiente.					
As variáveis usadas para filtrar são suficientes.					
A aplicação das variáveis é eficiente (filtros às não existentes).					

Com tecnologia
Google Forms

Appendix E

ID	Execution Time
1	953
2	887
3	849
4	849
5	856
6	869
7	896
8	887
9	873
10	866
11	872
12	853
13	857
14	858
15	878
16	893
17	887
18	865
19	865
20	858
21	872
22	903
23	859
24	882
25	890
26	880
27	861
28	848
29	877
30	859

Appendix F

MF (MAE)	MF (RMSE)	MF + UB (CC) (MAE)	MF + UB (CC) (RMSE)	MF + IB (MAE)	MF + IB (RMSE)	MF + UB + IB (MAE)	MF + UB + IB (RMSE)
0.780741337	1.020981976	1.544661649	1.778041811	0.776793576	0.978792897	1.3016349	1.543623
0.773678045	1.009705026	1.540897323	1.791600701	0.77670431	0.978295258	1.457158	1.466774
0.771248465	1.009431649	1.556980875	1.772536403	0.777882338	0.980018474	1.3854201	1.647687
0.773458628	1.010285927	1.540348599	1.788668079	0.778357426	0.981806943	1.5183734	1.513453
0.77253258	1.008379117	1.540547783	1.779759583	0.7775417	0.978533108	1.3679033	1.34467
0.777445994	1.013401576	1.544101545	1.784917521	0.774071039	0.973035406	1.3325993	1.679842
0.780634448	1.01848434	1.540551252	1.786001884	0.778093017	0.980253022	1.3015361	1.6453677
0.77409969	1.009143472	1.554931369	1.790697273	0.777994909	0.979216783	1.3864883	1.5675343
0.768428916	1.006784921	1.552707956	1.771527105	0.776057609	0.976731799	1.3280456	1.2356778
0.773762957	1.008885634	1.540518593	1.78018635	0.781457111	0.982319407	1.4517297	1.752372
0.774318279	1.013125248	1.544556291	1.787400763	0.777762771	0.979624833	1.469889	1.467832
0.775188441	1.012791834	1.551600196	1.792311101	0.780915133	0.983225495	1.4501857	1.584485
0.773376016	1.010853236	1.557600646	1.789161361	0.78100988	0.983115516	1.5723531	1.6988452
0.775024576	1.013839445	1.541110083	1.78662915	0.776534005	0.976245014	1.3434897	1.785162
0.776068194	1.015591295	1.551961751	1.778176009	0.778993117	0.980042137	1.3053905	1.782166
0.772797527	1.01091993	1.557188849	1.778479992	0.775079187	0.977167858	1.3559547	1.3614564
0.772310806	1.010548633	1.553033659	1.77333607	0.778720309	0.977399832	1.4307969	1.7548132
0.776740226	1.01661032	1.559333951	1.789102188	0.776492227	0.977337846	1.5593589	1.6515621
0.77662687	1.016914014	1.548349213	1.775831831	0.778684343	0.980221778	1.4450246	1.2325154
0.769370415	1.006815734	1.561049981	1.790941984	0.776288623	0.979081319	1.329739	1.5618456
0.775822298	1.012851253	1.548143423	1.779388579	0.77482572	0.975281767	1.5380591	1.4516515
0.772858831	1.012412051	1.562113436	1.77146569	0.780044065	0.98116758	1.5124173	1.7981512
0.775619304	1.015798338	1.546018884	1.77402286	0.777365829	0.980496679	1.3833593	1.6211454
0.770441523	1.010192947	1.54351397	1.78283881	0.777430801	0.980406109	1.4234124	1.611551
0.777423958	1.016942331	1.564757137	1.783454795	0.774533068	0.975698974	1.5219934	1.621854
0.767840614	1.004311801	1.5541732	1.781932085	0.776491581	0.977195824	1.4311004	1.65156
0.775743616	1.010901972	1.557993333	1.792319217	0.776858354	0.97713926	1.4225993	1.74164
0.771313941	1.008865803	1.558095772	1.776831276	0.777952702	0.980675164	1.4128309	1.75445
0.776263467	1.011504582	1.562568573	1.788955659	0.778291803	0.979612319	1.4481453	1.56145
0.777653532	1.014749656	1.549606908	1.775385618	0.780927807	0.982530945	1.4176105	1.62153

Appendix G

1 Feature			
index	mae	rmse-train	rmse-test
1	0.750165	0.762879114	0.954326243
2	0.749693	0.765482202	0.954673693
3	0.74695	0.765157726	0.952371983
4	0.744701	0.762217278	0.948882303
5	0.747244	0.768700446	0.952244622
6	0.747807	0.76395971	0.952046804
7	0.746408	0.765634166	0.950407901
8	0.749038	0.766317371	0.953193774
9	0.747735	0.764530994	0.952844736
10	0.747003	0.76208004	0.951372334
11	0.745344	0.767600543	0.950269103
12	0.746462	0.766813439	0.950907719
13	0.746792	0.76208116	0.951377378
14	0.748624	0.761029749	0.952295019
15	0.74542	0.7666325	0.951515329
16	0.749471	0.765442438	0.953897784
17	0.747897	0.763790818	0.953469492
18	0.748589	0.766884632	0.954862028
19	0.748234	0.764855902	0.953759671
20	0.751002	0.765618626	0.954999359
21	0.75051	0.76522037	0.955484947
22	0.753945	0.764960919	0.960586608
23	0.74437	0.7631352	0.950129836
24	0.745832	0.763116541	0.950271788
25	0.746747	0.764549733	0.951877201
26	0.748574	0.762850685	0.954024364
27	0.747876	0.766195142	0.953152859
28	0.745529	0.763068964	0.951157497
29	0.741617	0.763956186	0.946899284
30	0.747126	0.764867112	0.953984222

5 Features			
index	mae	rmse-train	rmse-test
1	0.760787	0.699145289	0.981585728
2	0.754733	0.700899211	0.975488764
3	0.751727	0.696325938	0.971280885
4	0.757874	0.691051773	0.977797572
5	0.753749	0.69805633	0.973677327
6	0.759216	0.696480676	0.980500187
7	0.760722	0.692258761	0.978128883
8	0.756195	0.695434093	0.977268316
9	0.755345	0.693847098	0.97658
10	0.761216	0.701404991	0.980739144
11	0.760363	0.692553285	0.979602888
12	0.761355	0.694729865	0.981048384
13	0.761783	0.694173062	0.982148163
14	0.756818	0.695296546	0.976946158
15	0.758724	0.69951807	0.976021535
16	0.760208	0.694889154	0.980032306
17	0.755763	0.688446509	0.975944952
18	0.76415	0.696926359	0.984389326
19	0.761369	0.694855834	0.979344345
20	0.759554	0.696201132	0.980422146

10 Features			
index	mae	rmse-train	rmse-test
1	0.768285	0.637304538	0.993332111
2	0.765672	0.631057573	0.994742016
3	0.765506	0.629606884	0.994582775
4	0.764452	0.63201591	0.991155135
5	0.760206	0.631624451	0.985824697
6	0.762496	0.631384116	0.994444988
7	0.757605	0.6305009	0.985399841
8	0.768512	0.630216199	0.996909786
9	0.768105	0.633495588	0.998165715
10	0.764364	0.6310457	0.991753339
11	0.763924	0.630453977	0.990018536
12	0.763103	0.629434886	0.989694157
13	0.768598	0.63037778	0.994123674
14	0.762803	0.631273018	0.992151027
15	0.758983	0.629416398	0.988772424
16	0.765277	0.63069136	0.992945785
17	0.760337	0.634105868	0.989144976
18	0.761114	0.629584391	0.98959046
19	0.761894	0.63284486	0.986070182
20	0.76578	0.631239902	0.995625766

15 Features			
index	mae	rmse-train	rmse-test
1	0.774883	0.582070176	1.007298999
2	0.773233	0.580093127	1.005963491
3	0.766795	0.580145135	0.999518528
4	0.765689	0.585325592	0.999885576
5	0.76821	0.582404268	1.000685243
6	0.768964	0.580333615	1.001157603
7	0.766298	0.587288711	1.000308138
8	0.76744	0.579577887	1.002727153
9	0.76821	0.590149239	0.999892386
10	0.768741	0.583477961	1.002308924
11	0.771296	0.578480056	1.004606356
12	0.766752	0.587245255	1.001672822
13	0.768328	0.582545163	1.003727172
14	0.769611	0.584138094	1.002958338
15	0.770415	0.579680993	1.003628157
16	0.766085	0.580493853	0.996413503
17	0.766667	0.583683091	1.000186771
18	0.761795	0.584067583	0.991062431
19	0.773115	0.579494893	1.00116571
20	0.773318	0.583502163	1.002967618

20 Features			
index	mae	rmse-train	rmse-test
1	0.773186	0.542282532	1.00932217
2	0.77976	0.539893327	1.019631397
3	0.768136	0.543611632	1.007131837
4	0.773256	0.540887765	1.009535347
5	0.777131	0.541283856	1.014239767
6	0.775926	0.542832951	1.012373581
7	0.773792	0.542977436	1.008158556
8	0.772637	0.541530932	1.010516938
9	0.774803	0.543180406	1.011176251
10	0.772957	0.542569727	1.010096698
11	0.775719	0.543486289	1.014203739
12	0.766959	0.543691226	1.006985801
13	0.771727	0.54404263	1.008237321
14	0.777807	0.538044945	1.017973624
15	0.777692	0.544579528	1.014299397
16	0.7695	0.543669814	1.006431742
17	0.77725	0.539979091	1.016695442
18	0.773338	0.546628947	1.010057407
19	0.771416	0.540555349	1.006319172
20	0.772455	0.542091644	1.006673833

30 Features			
index	mae	rmse-train	rmse-test
1	0.779641	0.479504279	1.024967357
2	0.783938	0.476512723	1.028693887
3	0.782461	0.480236516	1.021809102
4	0.782068	0.478547723	1.025949765
5	0.785714	0.479071656	1.025679165
6	0.786488	0.479085066	1.032418921
7	0.786198	0.479700663	1.02877403
8	0.782111	0.480737893	1.026801769
9	0.783676	0.480907393	1.027707268
10	0.784809	0.48039103	1.024515302
11	0.782041	0.479685843	1.027212708
12	0.788356	0.479020992	1.03011039
13	0.785927	0.480065317	1.029198017
14	0.78405	0.47838539	1.028321161
15	0.787317	0.476733314	1.028399962
16	0.792986	0.480252575	1.035103571
17	0.7849	0.480577543	1.02710611
18	0.790202	0.477236971	1.032636885
19	0.780893	0.478854355	1.02201556
20	0.78582	0.480871574	1.029956428

40 Features			
index	mae	rmse-train	rmse-test
1	0.801925	0.433386568	1.048006626
2	0.797409	0.434929885	1.044466521
3	0.800989	0.432282718	1.047077179
4	0.796813	0.431930096	1.047167695
5	0.800235	0.432658675	1.049476713
6	0.794691	0.43488617	1.041953125
7	0.801822	0.431908107	1.051323253
8	0.796764	0.432247026	1.043416391
9	0.792445	0.434020879	1.041077727
10	0.797181	0.429494488	1.050698491
11	0.795311	0.431482381	1.042356689
12	0.807389	0.431199078	1.056242523
13	0.801676	0.43234743	1.052606403
14	0.80114	0.431221333	1.04983089
15	0.796123	0.432633272	1.042330705
16	0.798955	0.430992525	1.047791627
17	0.798166	0.433949036	1.043113813
18	0.801044	0.42978128	1.050207826
19	0.792775	0.431544242	1.041649561
20	0.797007	0.429671257	1.044077824

Appendix H

Matrix Factorization (500 epochs)	
Epoch	Error (RMSE)
1	1.086042326
2	0.93899721
3	0.89294286
4	0.867278025
5	0.849487461
6	0.835851189
7	0.824751972
8	0.815336417
9	0.807095066
10	0.799700409
11	0.792931768
12	0.786635276
13	0.780700603
14	0.77504676
15	0.76961319
16	0.764354071
17	0.75923456
18	0.754228242
19	0.749315309
20	0.744481212

21	0.739715609
22	0.735011527
23	0.730364669
24	0.725772836
25	0.721235434
26	0.716753062
27	0.712327158
28	0.707959704
29	0.703652989
30	0.699409414
31	0.695231338
32	0.691120973
33	0.687080296
34	0.683111005
35	0.679214485
36	0.675391801
37	0.671643697
38	0.667970609
39	0.664372691
40	0.660849831
41	0.65740168
42	0.654027684
43	0.650727103
44	0.647499043
45	0.644342476

46	0.641256265
47	0.638239181
48	0.635289927
49	0.632407146
50	0.62958944
51	0.626835383
52	0.624143529
53	0.621512421
54	0.618940601
55	0.616426615
56	0.613969019
57	0.611566385
58	0.609217301
59	0.60692038
60	0.604674256
61	0.602477591
62	0.600329075
63	0.598227425
64	0.596171392
65	0.594159753
66	0.59219132
67	0.590264932
68	0.588379462
69	0.586533814
70	0.584726921

71	0.582957747
72	0.581225286
73	0.579528562
74	0.577866627
75	0.57623856
76	0.574643471
77	0.573080493
78	0.571548787
79	0.570047541
80	0.568575965
81	0.567133295
82	0.565718789
83	0.564331729
84	0.562971419
85	0.561637183
86	0.560328367
87	0.559044337
88	0.557784477
89	0.556548192
90	0.555334902
91	0.554144048
92	0.552975085
93	0.551827486
94	0.55070074
95	0.54959435

96	0.548507835
97	0.547440728
98	0.546392576
99	0.545362937
100	0.544351386
101	0.543357507
102	0.542380898
103	0.541421168
104	0.540477936
105	0.539550834
106	0.538639503
107	0.537743594
108	0.536862769
109	0.535996699
110	0.535145064
111	0.534307552
112	0.53348386
113	0.532673695
114	0.531876769
115	0.531092804
116	0.530321529
117	0.52956268
118	0.528815999
119	0.528081237
120	0.527358149

121	0.526646498
122	0.525946054
123	0.52525659
124	0.524577888
125	0.523909732
126	0.523251916
127	0.522604235
128	0.521966491
129	0.521338491
130	0.520720047
131	0.520110974
132	0.519511093
133	0.518920229
134	0.518338211
135	0.517764873
136	0.517200051
137	0.516643586
138	0.516095324
139	0.515555113
140	0.515022805
141	0.514498254
142	0.513981321
143	0.513471866
144	0.512969755
145	0.512474856

146	0.511987041
147	0.511506182
148	0.511032157
149	0.510564846
150	0.510104129
151	0.509649894
152	0.509202025
153	0.508760414
154	0.508324952
155	0.507895534
156	0.507472056
157	0.507054417
158	0.506642518
159	0.506236262
160	0.505835554
161	0.505440301
162	0.505050413
163	0.504665799
164	0.504286372
165	0.503912048
166	0.503542742
167	0.503178371
168	0.502818856
169	0.502464118
170	0.502114078

171	0.501768662
172	0.501427796
173	0.501091405
174	0.500759419
175	0.500431769
176	0.500108384
177	0.499789198
178	0.499474145
179	0.49916316
180	0.498856179
181	0.49855314
182	0.498253982
183	0.497958645
184	0.49766707
185	0.497379199
186	0.497094976
187	0.496814344
188	0.49653725
189	0.496263639
190	0.495993458
191	0.495726657
192	0.495463185
193	0.495202991
194	0.494946026
195	0.494692244

196	0.494441595
197	0.494194035
198	0.493949518
199	0.493707998
200	0.493469433
201	0.493233778
202	0.493000992
203	0.492771033
204	0.492543859
205	0.492319432
206	0.492097711
207	0.491878658
208	0.491662234
209	0.491448402
210	0.491237126
211	0.491028369
212	0.490822095
213	0.49061827
214	0.490416859
215	0.490217828
216	0.490021145
217	0.489826777
218	0.489634691
219	0.489444856
220	0.489257241

221	0.489071816
222	0.48888855
223	0.488707415
224	0.48852838
225	0.488351419
226	0.488176502
227	0.488003602
228	0.487832691
229	0.487663744
230	0.487496734
231	0.487331636
232	0.487168423
233	0.487007071
234	0.486847555
235	0.486689851
236	0.486533935
237	0.486379785
238	0.486227376
239	0.486076686
240	0.485927694
241	0.485780376
242	0.485634712
243	0.485490679
244	0.485348258
245	0.485207427

246	0.485068167
247	0.484930457
248	0.484794278
249	0.484659609
250	0.484526434
251	0.484394731
252	0.484264484
253	0.484135674
254	0.484008282
255	0.483882292
256	0.483757686
257	0.483634447
258	0.483512558
259	0.483392002
260	0.483272763
261	0.483154826
262	0.483038173
263	0.48292279
264	0.48280866
265	0.48269577
266	0.482584104
267	0.482473647
268	0.482364384
269	0.482256302
270	0.482149386

271	0.482043623
272	0.481938998
273	0.481835499
274	0.481733111
275	0.481631822
276	0.481531619
277	0.481432489
278	0.481334419
279	0.481237397
280	0.481141411
281	0.481046449
282	0.480952499
283	0.480859549
284	0.480767588
285	0.480676603
286	0.480586585
287	0.480497521
288	0.480409402
289	0.480322215
290	0.480235951
291	0.480150598
292	0.480066147
293	0.479982588
294	0.479899909
295	0.479818102

296	0.479737156
297	0.479657062
298	0.479577809
299	0.47949939
300	0.479421793
301	0.479345011
302	0.479269034
303	0.479193852
304	0.479119458
305	0.479045842
306	0.478972996
307	0.478900911
308	0.478829579
309	0.478758991
310	0.47868914
311	0.478620016
312	0.478551613
313	0.478483922
314	0.478416936
315	0.478350646
316	0.478285045
317	0.478220126
318	0.478155881
319	0.478092302
320	0.478029383

321	0.477967117
322	0.477905496
323	0.477844513
324	0.477784162
325	0.477724435
326	0.477665326
327	0.477606828
328	0.477548935
329	0.477491639
330	0.477434936
331	0.477378818
332	0.477323278
333	0.477268312
334	0.477213912
335	0.477160073
336	0.477106788
337	0.477054052
338	0.477001859
339	0.476950204
340	0.476899079
341	0.47684848
342	0.476798402
343	0.476748837
344	0.476699783
345	0.476651231

346	0.476603179
347	0.476555619
348	0.476508547
349	0.476461959
350	0.476415847
351	0.476370209
352	0.476325038
353	0.47628033
354	0.47623608
355	0.476192283
356	0.476148934
357	0.476106028
358	0.476063562
359	0.47602153
360	0.475979928
361	0.475938751
362	0.475897995
363	0.475857655
364	0.475817728
365	0.475778208
366	0.475739091
367	0.475700374
368	0.475662052
369	0.475624121
370	0.475586577

371	0.475549416
372	0.475512634
373	0.475476227
374	0.475440191
375	0.475404522
376	0.475369216
377	0.47533427
378	0.47529968
379	0.475265442
380	0.475231553
381	0.475198009
382	0.475164806
383	0.47513194
384	0.475099409
385	0.475067209
386	0.475035337
387	0.475003788
388	0.47497256
389	0.47494165
390	0.474911054
391	0.474880768
392	0.47485079
393	0.474821117
394	0.474791745
395	0.474762671

396	0.474733892
397	0.474705406
398	0.474677208
399	0.474649297
400	0.474621669
401	0.474594321
402	0.474567251
403	0.474540455
404	0.474513931
405	0.474487675
406	0.474461686
407	0.47443596
408	0.474410495
409	0.474385288
410	0.474360337
411	0.474335638
412	0.474311189
413	0.474286988
414	0.474263032
415	0.474239319
416	0.474215845
417	0.47419261
418	0.474169609
419	0.474146841
420	0.474124304

421	0.474101995
422	0.474079911
423	0.474058051
424	0.474036411
425	0.474014991
426	0.473993787
427	0.473972798
428	0.473952021
429	0.473931454
430	0.473911094
431	0.473890941
432	0.473870991
433	0.473851243
434	0.473831694
435	0.473812343
436	0.473793187
437	0.473774225
438	0.473755454
439	0.473736873
440	0.47371848
441	0.473700272
442	0.473682248
443	0.473664406
444	0.473646744
445	0.47362926

446	0.473611953
447	0.47359482
448	0.47357786
449	0.473561072
450	0.473544452
451	0.473528001
452	0.473511715
453	0.473495594
454	0.473479635
455	0.473463837
456	0.473448198
457	0.473432717
458	0.473417392
459	0.473402222
460	0.473387205
461	0.473372339
462	0.473357622
463	0.473343054
464	0.473328633
465	0.473314357
466	0.473300225
467	0.473286236
468	0.473272387
469	0.473258678
470	0.473245107

471	0.473231673
472	0.473218373
473	0.473205208
474	0.473192176
475	0.473179274
476	0.473166503
477	0.47315386
478	0.473141344
479	0.473128955
480	0.47311669
481	0.473104548
482	0.473092529
483	0.473080631
484	0.473068852
485	0.473057192
486	0.47304565
487	0.473034223
488	0.473022911
489	0.473011714
490	0.473000629
491	0.472989655
492	0.472978792
493	0.472968038
494	0.472957392
495	0.472946853

496	0.47293642
497	0.472926092
498	0.472915868
499	0.472905747
500	0.472895728

Appendix I

Tested Method	*CompareVectors(userfeaturesOne, userFeaturesTwo)*
Method Location	Fitness Intelligent Recommender - CoRatedCosineUserComparer class
Method Description	Compares two user feature vectors and returns the degree of similarity.
Tests	• Send valid vectors • Send invalid null vectors
Result	Every test passed.

Tested Method	*CompareVectors(userfeaturesOne, userFeaturesTwo)*
Method Location	Fitness Intelligent Recommender - CorrelationUserComparer class
Method Description	Compares two user feature vectors and returns the degree of similarity.
Tests	• Send valid vectors • Send invalid null vectors
Result	Every test passed.

Tested Method	*CompareVectors(userfeaturesOne, userFeaturesTwo)*
Method Location	Fitness Intelligent Recommender - CosineUserComparer class
Method Description	Compares two user feature vectors and returns the degree of similarity.
Tests	• Send valid vectors • Send invalid null vectors
Result	Every test passed.

Tested Method	*CompareVectors(userfeaturesOne, userFeaturesTwo)*
Method Location	Fitness Intelligent Recommender - RootMeanSquareComparer class
Method Description	Compares two user feature vectors and returns the degree of similarity.
Tests	• Send valid vectors • Send invalid null vectors
Result	Every test passed.

Tested Method	*GenerateLinearBestFit(points)*
Method Location	Fitness Intelligent Recommender - LinearBestFit class
Method Description	Calcualtes the line of best fit based on a set of points.
Tests	• Send valid points • Send invalid null vectors
Result	Every test passed.

Tested Method	*GetSlope(points)*
Method Location	Fitness Intelligent Recommender - LinearBestFit class
Method Description	Calculates the slope of a line based on its points.
Tests	• Send valid points • Send invalid null vectors
Result	Every test passed.

Tested Method	*GetPredictedValueForX(points, slope, x)*
Method Location	Fitness Intelligent Recommender - LinearBestFit class
Method Description	Calculates the intersection in the Y-axis of the line of best fit, based on the X-value.

Tests	• Send valid points and slope • Send valid points and invalid slope • Send invalid points and slope • Send valid x • Send invalid x
Result	Every test passed.

Tested Method	*GetDotProduct(vectorone, vectortwo)*
Method Location	Fitness Intelligent Recommender - Matrix class
Method Description	Calculates dot products between two vectors.
Tests	• Send valid vectors • Send invalid vectors
Result	Every test passed.

Tested Method	*FactorizeMatrix(matrix)*
Method Location	Fitness Intelligent Recommender - SingularValueDecomposition class
Method Description	Calculates the svd for a matrix.
Tests	• Send valid matrix • Send invalid matrix
Result	Every test passed.

Tested Method	*LoadData()*
Method Location	Fitness Intelligent Recommender - ImportData class
Method Description	Imports the dataset.
Tests	• Use valid dataset

	• Use invalid dataset
Result	Every test passed.

Tested Method	*Train(datasetmodel)*
Method Location	Fitness Intelligent Recommender - ItemCollaborativeFilterRecommender class
Method Description	Trains the model.
Tests	• Send a valid model • Send an invalid model
Result	Every test passed.

Tested Method	*GetRating(userID, itemID)*
Method Location	itness Intelligent Recommender - ItemCollaborativeFilterRecommender class
Method Description	Gets the rating of a given user for a given item.
Tests	• Use valid user and item • Use valid user and invalid item • Use invalid user and valid item • Use invalid user and invalid item
Result	Every test passed.

Tested Method	*GetSuggestions(userID, numberOfSuggestions)*
Method Location	itness Intelligent Recommender - ItemCollaborativeFilterRecommender class
Method Description	Gets suggestions for a given user.
Tests	• Use valid user and number of suggestions • Use valid user and invalid number of suggestions

	• Use invalid user and valid number of suggestions • Use invalid user and invalid number of suggestions
Result	Every test passed.

Tested Method	*Train(datasetmodel)*
Method Location	Fitness Intelligent Recommender - MatrixFactorizationRecommender class
Method Description	Trains the model.
Tests	• Send a valid model • Send an invalid model
Result	Every test passed.

Tested Method	*GetRating(userID, itemID)*
Method Location	itness Intelligent Recommender - MatrixFactorizationRecommender class
Method Description	Gets the rating of a given user for a given item.
Tests	• Use valid user and item • Use valid user and invalid item • Use invalid user and valid item • Use invalid user and invalid item
Result	Every test passed.

Tested Method	*GetSuggestions(userID, numberOfSuggestions)*
Method Location	itness Intelligent Recommender - MatrixFactorizationRecommender class
Method Description	Gets suggestions for a given user.
Tests	• Use valid user and number of suggestions

	• Use valid user and invalid number of suggestions • Use invalid user and valid number of suggestions • Use invalid user and invalid number of suggestions
Result	Every test passed.

Tested Method	*Train(datasetmodel)*
Method Location	Fitness Intelligent Recommender - UserCollaborativeFilterRecommender class
Method Description	Trains the model.
Tests	• Send a valid model • Send an invalid model
Result	Every test passed.

Tested Method	*GetRating(userID, itemID)*
Method Location	itness Intelligent Recommender - UserCollaborativeFilterRecommender class
Method Description	Gets the rating of a given user for a given item.
Tests	• Use valid user and item • Use valid user and invalid item • Use invalid user and valid item • Use invalid user and invalid item
Result	Every test passed.

Tested Method	*GetSuggestions(userID, numberOfSuggestions)*
Method Location	itness Intelligent Recommender - UserCollaborativeFilterRecommender class
Method Description	Gets suggestions for a given user.

Tests	• Use valid user and number of suggestions • Use valid user and invalid number of suggestions • Use invalid user and valid number of suggestions • Use invalid user and invalid number of suggestions
Result	Every test passed.

Tested Method	*filterVolumeAndIntensity(recommendations, predictedvolume, predicted intensity)*
Method Location	Fitness Conditional Recommender
Method Description	Filters the tecommendation list based on the predicted volume and intensity.
Tests	• Send valid predicted values • Send invalid predicted values
Result	Every test passed.

Tested Method	*predictIntensity(intensityListy)*
Method Location	Fitness Conditional Recommender
Method Description	Predicts the intensity based on the previous intensity list.
Tests	• Send valid intensity list • Send invalid intensity list
Result	Every test passed.

Tested Method	*predictVolume (volumeList)*
Method Location	Fitness Conditional Recommender
Method Description	Predicts the volume based on the previous volume list.
Tests	• Send valid volume list

	• Send invalid volume list
Result	Every test passed.

Appendix J

The tests here presented are being generalized to all the tests under that specific type. For example "*Gets*" includes all GET requests. These include the requests from the following entities: Body Measurements, Equipment, Equipment Categories, Programs, Plans.

Tested Method	*Get()*
Involved Components	*FitnessMobileAPP, FitnessAPI,* Database
Description	The mobile app asks the *API* to retrieve some entity/entity list. The *API* asks the Database for the desired information.
Expected Result	The Database returns the desired entity/entity list.
Result	Success.

Tested Method	*Post()*
Involved Components	*FitnessMobileAPP, FitnessAPI,* Database
Description	The mobile app asks the *API* to save new data. The *API* requests the Database to register the new data.
Expected Result	The Database saves the information and returns the operation's success.
Result	Success.

Tested Method	*Put()*
Involved Components	*FitnessMobileAPP, FitnessAPI,* Database
Description	The mobile app asks the *API* to edit data. The *API* requests the Database to change the data.
Expected Result	The Database saves the information and returns the operation's success.
Result	Success.

The following tests are relative to the chat bot aspect of the system.

Tested Method	*GetIntentScore()*
Involved Components	*ChatBotAPI, LUISAPI*
Description	The *ChatBotAPI* requests the *LUISAPI* to retrieve the score associated with a specific intent.
Expected Result	The *LUISAPI* returns the correct score.
Result	Success.

Tested Method	*Get()*
Involved Components	*ChatBotAPI, FitnessAPI*, Database
Description	The *FitnessAPI* obtains the exercise data from the Database and returns it to the *ChatBotAPI*
Expected Result	The Database returns the exercise list and the *ChatBotAPI* receives it correctly.
Result	Success.

Tested Method	*GetResponse()*
Involved Components	*FitnessMobileAPP, FitnessAPI, ChatBotAPI*, *LUISAPI*, Database
Description	The *FitnessMobileAPP* receives a message from the user, sends it to the *FitnessAPI* which redirects it to the *ChatBotAPI* which gets its intent from the *LUISAPI. The* Database is used by the *FitnessAPI* if necessary. The response is then retrieved to the mobile app.
Expected Result	The response is correctly formulated.
Result	Success.

Appendix K

Description	The user creates a Program, opens the Programs list and views its details.
Result	Success.

Description	The user creates a Plan, opens the Programs list and views its details.
Result	Success.

Description	The user configures their body measurements, adds specific goals, adds logs and views the progress monitoring.
Result	Success.

Description	The user edits their preferred body measurements.
Result	Success.

Description	The user changes the system's language and views.
Result	Success.

Description	The user opens the calendar and selects the desired day to view its logs.
Result	Success.

Description	The user changes the available equipment and refreshes the screen to see the updates.
Result	Success.

Appendix L

Tested Use Case	US02: View training Plan/Program templates
Expected Result	The correct Plan/Program template list is shown.
Result	Success.

Tested Use Case	US03: Manage training Plan/Program
Expected Result	The new Plan/Program is created and correctly added to both the mobile app and the database.
Result	Success.

Tested Use Case	US04: Manage Body Measurements
Expected Result	The configured body measurements are correctly shown.
Result	Success.

Tested Use Case	US04: Manage Body Measurements
Expected Result	The added logs are correctly shown in their respective body measurements.
Result	Success.

Tested Use Case	US04: Manage Body Measurements
Expected Result	The corrected re-configured body measurements are correctly shown.
Result	Success.

Tested Use Case	US08: View training Logs
Expected Result	The correct logs are shown.

Result	Success.

Tested Use Case	US010: View Statistics
Expected Result	The correct statistics are shown.
Result	Success.

Tested Use Case	US015: Change Language
Expected Result	The new language is saved, and the app is updated.
Result	Success.

Tested Use Case	US016: Change Unit System
Expected Result	The new unit system is saved, and the app updated.
Result	Success.

Tested Use Case	US017: Manage Available Equipment
Expected Result	The configured available equipment are correctly shown.
Result	Success.

Tested Use Case	US017: Manage Available Equipment
Expected Result	The re-configured available equipment are correctly shown.
Result	Success.