INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO

MESTRADO EM ENGENHARIA INFORMÁTICA

Sistema de recomendação para alívio de stress

FRANCISCO MIGUEL MAGALHÃES PINELAS

Outubro de 2022

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

Recommendation System to Avoid Stress

Francisco Miguel Magalhães Pinelas

Dissertation for obtaining the Master's Degree in Informatics Engineering, Area of Specialization in Information and Knowledge Systems

Supervisor: Fátima Rodrigues

Porto, October 2022

Abstract

Stress is considered to be a normal part of our lives, especially when taken into account that people are constantly trying to push their limits and the limits of others around them. Whether at home or at their jobs, the idea that to be successful one must work harder is deeply rooted within society because such behavior has shown positive results in the past. Stressful events can work as a reactor for people to feel the necessary motivation to move on with their tasks. However, if uncontrolled, may lead to health-related consequences, such as cardiovascular diseases, sleep deprivation, and anxiety. Therefore, it is important to not only recognize that stress has serious negative impacts in the lives of people but also to find mechanisms to cope with it.

This document presents a solution in the form of a web application capable of providing stress-easing and health improving recommendations that adapt to the users of the application by considering their ratings from past recommendation, as well as their profiles. Through an engaging and interactive graphical user interface, users can receive personalized recommendations via system notifications. These notifications are composed of a demonstrative card and a description, as well as a collection of documents with insightful information regarding the recommendations being provided.

This web application is supported by three major components, designed to operate both synchronously and asynchronously in a microservices-oriented architecture, promoting the flexibility and scalability of the solution.

Furthermore, for the solution to provide recommendations, it was necessary to implement a filtering technique. Among the most common ones, the content-based filtering is the most advantageous, meaning that a content-based recommender system was developed as part of the solution.

Lastly, it was concluded that the web application satisfactorily meets the established requirements. However, due to lack of user-generated data, the randomly generated data used to demonstrate how a proper evaluation would be conducted cannot be subject for interpretation.

Keywords: Stress, Recommender system, Recommendations, Filtering techniques, Content-based filtering

Resumo

O stress é considerado parte integrante nas nossas vidas, especialmente quando consideramos que as pessoas estão constantemente a tentar superar os seus limites, muitas vezes delegando essas mesmas expectativas àqueles que as rodeiam. Tanto em casa, com os seus parceiros e familiares, como no seu local de trabalho, a ideia de que para se obter sucesso tem de se trabalhar mais está enraizada na sociedade, sendo que comportamentos semelhantes no passado comprovaram resultados positivos nesse sentido. Os eventos de stress podem funcionar como um sentimento de motivação para que as pessoas consigam desempenhar as suas funções diariamente. Contudo, caso não sejam controlados, poderão trazer consequências graves relacionadas com a saúde, tais como doenças cardiovasculares, privação do sono, e problemas de ansiedade. Assim, é importante não apenas reconhecer que o stress tem diversos impactos sérios na vida das pessoas, mas também encontrar formas de o gerir apropriadamente.

Este documento apresenta uma solução na forma de uma aplicação *web* capaz de fornecer recomendações para redução do stress e melhoria de saúde, que se adaptam aos utilizadores da aplicação considerando as suas avaliações a recomendações no passado. Através de uma interface gráfica envolvente e interativa, os utilizadores podem receber recomendações personalizadas por meio de notificações do sistema. Essas notificações são compostas por um cartão demonstrativo e uma descrição, assim como de um conjunto de documentos com informações detalhadas acerca das recomendações fornecidas.

Esta aplicação é composta por três componentes principais, desenhados para interagirem entre si tanto de forma síncrona como assíncrona numa arquitetura orientada a microserviços, promovendo a flexibilidade e escalabilidade da solução.

Para além disso, para que a solução seja capaz de fornecer recomendações, foi preciso implementar uma técnica de filtragem. Entre as mais comuns, a filtragem baseada em conteúdo demonstrou ser a mais vantajosa, significando que foi desenvolvido um sistema de recomendação baseado em conteúdo como parte da solução.

Finalmente, concluiu-se que a aplicação *web* vai satisfatoriamente de encontro aos requisitos estabelecidos. Contudo, devido à falta de dados gerados pelos utilizadores da aplicação, os dados gerados aleatoriamente para demonstrar de que forma uma avaliação adequada seria realizada, não estão sujeitos a interpretação.

Palavras-chave: Stress, Sistema de recomendação, Recomendações, Técnicas de filtragem, Filtragem baseada em conteúdo

Acknowledgements

I would like to offer my most sincere thanks to my supervisor Fátima Rodrigues for providing me with this opportunity to be part of a project within the scope of an H2020 initiative, but also for the indispensable support and guidance throughout its duration. From regular meetings and exchange of e-mails outside normal working hours due to my current professional situation, to the anticipation of relevant aspects that led to the success of the project, I felt that I received all the necessary support to get through this challenge. For that I am truly thankful.

I would also like to thank Matilde Rodrigues, Nuno Rocha, and Simão Ferreira from Escola Superior de Saúde as the stakeholders for this project. Their health-related expertise has shown crucial to the development of this project.

To the colleagues and friends that I met along this journey on ISEP, who helped molding my experience and contributed to my success in many ways.

To my work colleagues at KPMG, who supported my decision to put my academic course in first place and have shown flexibility regarding my working hours.

Finally, I would like to give my deepest gratitude to my family, especially my parents and sister, but also to Margarida Dias, with whom I shared this entire journey with, along with its sacrifices and successes.

Index

Figure Index

Table Index

Code Snippets Index

1 Introduction

This chapter aims to present a theoretical framework of the project to be developed in the area of Recommender Systems related to stress.

It begins by contextualizing the project, proceeding with a description on the problem and the objective behind its elaboration.

Furthermore, it is presented an approach on how to address the main goal of this project and exposed the expected results, ending with the structure of this document.

1.1 Context

As modern society becomes more complex and digitalization takes over the economy, so increases the number of workers that suffer from mental health disorders such as stress.

Stress is often defined as the human response to pressure, and it is usually triggered when under the presence of some negative experience, whether unexpected or not, especially when accompanied with the feeling of lack of control over the situation.

Proper and regulated stress at work allows one to immerse in the task at hand, serving almost as fuel to increase productivity in the workplace, resulting in overall satisfaction. A healthy and motivated workforce is the key to success. However, excessive stress or long-term stress at work, often lead to occupational burn-out among other lasting impacts in our health.

Mad@Work[1] states that stress costs employers a great deal both directly (up to 50% of all lost working days are linked to stress) and indirectly (by reducing work engagement). In the

[1] Mental Wellbeing Management and Productivity Boosting in the Workplace (Mad@Work), POCI-01-0247-FEDER-046168, Sistema de Incentivos à Investigação e Desenvolvimento Tecnológico (SI&IDT), 09/2020 a 06/2023. https://itea3.org/project/mad-work.html accessed on 22/01/2022

European Union alone, 24.4% of workers experience frequent or constant stress at work. This, in turn, also has a monetary impact, resulting in an estimated loss of €617 billion annually.

As companies realize the impact of a safe and balanced workplace, as well as mentally healthy workers, and how such environments lead to better results and overall success in the long-term, efforts are being made to provide a compelling working experience and ways to avoid and overcome intense stress related situations.

1.2 Problem

Stress at work has become a serious problem affecting many people of different professions, life situations, and age groups, and continually contributes to illness either directly, through its physiological effects such as cardiovascular diseases and anxiety, or indirectly, through bad health behaviors such as lack of sleep.

However, sometimes people lack the self-awareness to realize that they are under an episode of stress and will much faster occupy their minds with tasks at hand such as a very demanding job, or even more mundane things like looking for something new to wear, places to visit, or shows to watch, rather than reflecting upon their well-being and how much of an impact it is suffering from their day-to-day activities.

Stress is not palpable, and it is hard to visualize, takes many different shapes within society, is mostly unpredictable because it is very personal and is dependent on the context of every individual. Constantly being pushed to achieve more and go farther, stress mostly goes unnoticed until people start feeling the consequences, like the ones mentioned above.

Mechanisms to provide awareness and aid during and after episodes of stress could serve as a way to mitigate its side effects.

There have been developed solutions in the past to address similar matters through the usage of recommendations, however often overlooking the importance of explaining the reason behind such recommendations and the benefits from acting upon the described actions (Croon, Houdt, Htun, Stiglic, Abeele, & Verbert 2021).

1.3 Objective

It is important to motivate people to use appropriate stress coping strategies in order to adjust their behavior and lifestyle so that they achieve a better stress balance and avoid increased level of stress that may result in serious health problems.

The main goal for this project is to develop a Recommender System, integrated in a web application, as part of an H2020 initiative, in conjunction with Escola Superior de Saúde from Instituto Politécnico do Porto, related to the detection of stress in the workplace in a non-evasive way.

By integrating with a machine learning model capable of analyzing stress indicators, external to the scope of this project, this recommendation system should be based on pre-defined rules and be capable of suggesting to users some of the most effective and practical actions that they could take in order to lower the levels of stress that they may be experiencing, by considering their characteristics and preferences.

1.4 Approach

To propose a solution capable of addressing the problem described in subchapter 1.2, several meetings were conducted with the client, in this case being Escola Superior de Saúde. This way, it was possible to understand the main purpose behind the project, to define the necessary requirements, but also to manage expectations.

Then, having a solid objective for the problem at hand, it was conducted a state of the art about recommender systems within the context of stress, including the types of filtering techniques that are commonly used.

After concluding the literature analysis, the value proposition was described, where the issues and pains of the target audience are exposed, as well as the benefits behind the objective of this project and how it aims to address the problems of the target audience. Furthermore, it is justified the selection of the filtering technique to be used based on the definition of criteria and using scientific methods of selection.

Then, considering the requirements of the client and the state of the art, the analysis and design on which the development of the solution would be based was carried out, detailing every component within it, and the implementation conducted afterwards.

Lastly, to prove that the solution delivers positive results, it was defined a hypothesis, metrics, and methodologies upon which to conduct a system evaluation. Ideally, this evaluation would be based on real data. However, due to delivery constraints related to the involvement of other parties in the project, it was used randomly generated data.

1.5 Expected Results

This document aims to describe all the steps involved in the development of an application capable of providing its users with adequate and personalized recommendations related to stress easing actions, while also increasing awareness of the impact that stress has on their day-to-day lives and the benefits associated with maintaining stress levels within their control.

The application is expected to have an appealing and interactive graphical interface to the end users, through which they will be able to obtain and rate tailored health-related recommendations, as the means to promote a healthier and more conscious lifestyle.

1.6 Document Structure

This document is structured in seven chapters.

The first chapter presents the context of the matter, as well as the problem being addressed, and the necessary steps to achieve the desired goal.

The second chapter describes the definition of a recommender system and the different types of filtering techniques that are often applied within one. In this chapter some existing solutions related to the development of a recommender system within the scope of stress are also presented.

The third chapter presents both the value analysis and the proposal analysis.

The fourth chapter presents the analysis and design of the solution, including functional and non-functional requirements, use cases, domain model, and an architectural proposal.

The fifth chapter describes the implementation of the solution based on the proposal presented in the fourth chapter.

The sixth chapter presents a critical analysis of the obtained results from the implementation of the solution, and the seventh chapter finishes with a conclusion, where achieved goals and future work are also mentioned.

2 State of the Art

This chapter begins with a description about Recommender Systems, the motives behind their creation, and what they are currently used for.

Considering that the core of a Recommender System is the underlying filtering technique, the ones that are mostly used are also described in this chapter along with their strengths and weaknesses.

This chapter ends with an overview about how these filtering techniques are technically approached, followed by some of the most relevant work related to the integration of a Recommender System in the context of stress relieving solutions.

2.1 Introduction

Recommender Systems, or Recommendation Systems, are software that implement specific algorithms with the sole purpose of making suggestions, within the available options for a given context, that a user may find relevant.

Ever since the arise of Web 2.0, that is often referred to as the era in which worldwide websites began to highlight user-generated content, usability, and interoperability (Hiremath, 2016), the information available on the internet started to grow exponentially.

Having access to all that information a few clicks away is undoubtedly both inspiring and empowering. However, the pace at which the data keeps being generated brought many challenges, one of them being how to properly use and make the most of it.

In a society that favors having an experience tailored to its personal and collective needs, it became apparent that a systematic way to filter the information that is presented to the users was necessary. Studies regarding consumer over choice and decision paralysis have been conducted over the years, concluding that when a consumer is faced with an overwhelming number of alternatives, sometimes fails to analyze them all and even leads to the option of not making a choice at all (Rosen & Boccia 2021).

Recommender Systems are present to aid the users of an application or service in their decision-making process on which action to engage in next and which information to consume.

There are multiple types of Recommender Systems, as described in detail throughout the next subchapters, and they all differ in terms of architecture. However, a general overview of a Recommender System architecture can be designed as shown in Figure 1:

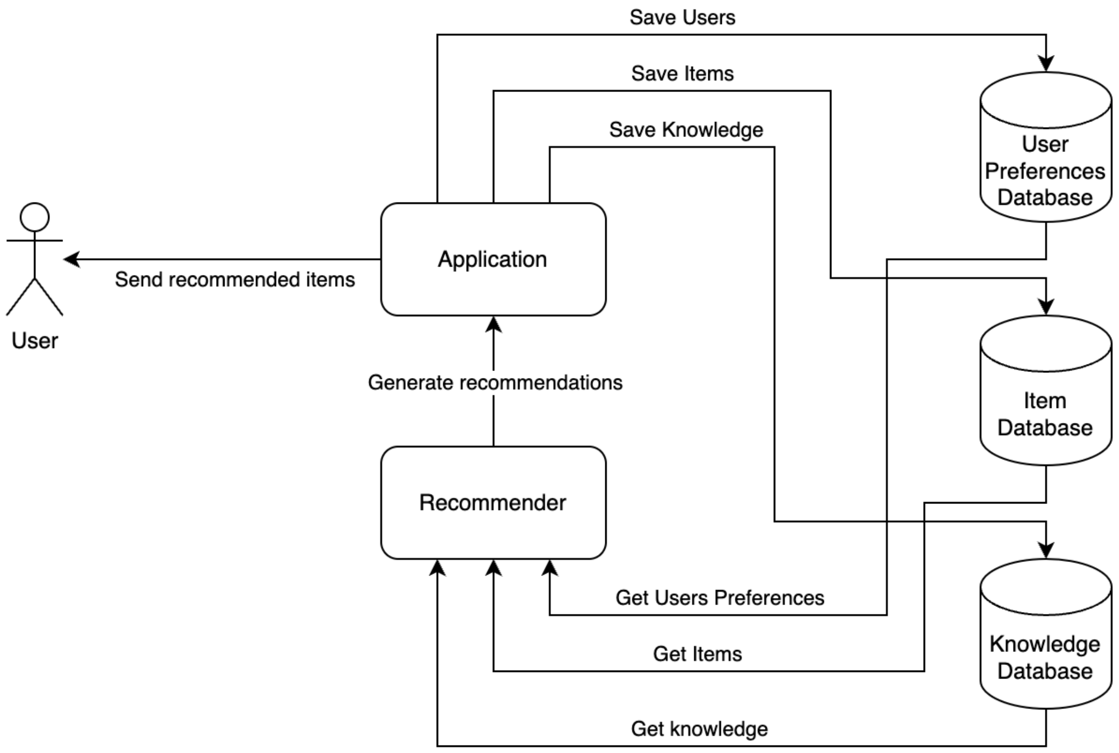


Figure 1 - General architecture of a recommender system

2.2 Filtering Techniques

The recommendations from Recommender Systems are made based on different filtering techniques, the most used ones being Collaborative filtering, Content-based filtering, and Knowledge-based filtering. Often, a Recommender System is classified after one type of filtering. However, there are cases where Recommender Systems benefit the most by applying more than one filtering technique, resulting in Hybrid Recommender Systems.

2.2.1 Collaborative Filtering

The Collaborative Filtering technique is based on the idea that users who have agreed in the past regarding an evaluation of certain items are likely to agree with the evaluation of other items again in the future, meaning that in this type of filtering is done collaboratively having user preferences in consideration.

Collaborative Filtering is applied in many Recommender Systems throughout the many platforms and services that we have at our disposal over the internet in fields such as e-commerce, marketing, social networks, video streaming, and customer relationship management (Mustafa, Ibrahim, Ahmed, & Abdullah 2017).

This type of filtering can generally be approached either by a memory-based method or a model-based method:

- Memory-based method

  Memory-based method, also known as Neighborhood-based or Heuristic-based method, operates over the entire database of user-generated ratings on older data to predict the unknown rating of other items by users with great similarity in terms of interests.

  This method, in turn, can be approached in a User-based or Item-based way.

  The User-based approach evaluates the interest of a user in a specific item by having in consideration the ratings of that item from other users with similar rating patterns of the user.

  The Item-based approach evaluates the interest of a user in a specific item by having in consideration the ratings of the user for similar items.

- Model-based method

  Instead of using ratings stored in a database to predict new ones in a User-based or Item-based approach, this method takes the user-generated ratings to acquire knowledge and learn a predictive model capable of replicating the interaction between users and items while having in consideration some more complex aspects such as user preferences and item categories. This model is then used to predict the ratings of users for new items.

  Model-based methods usually rely on Clustering, Latent Semantic Analysis, Support Vector Machines, and Singular Value Decomposition (Shah & Salunke 2017).

Implementing a Collaborative Filtering technique does not require domain knowledge of the items, the quality of the recommendations increase over time, the recommendations are personalized, and allows for users to discover new interests. However, the quality of the recommendations is highly dependent on large historical data sets, does not scale well, there is usually a lot of sparsity in the data since only small subsets of the items are rated by the users, and lacks the desired performance for cases when a new user or item is added to the database, often referred to as the cold-start problem.

2.2.2 Content-based Filtering

Content-based Filtering, also known as Cognitive Filtering (Shah & Salunke 2017), is another widely adopted filtering technique that selects items based on the correlation between the content of the items and the user preferences or items that the user liked in the past. It is based on the concept that items with similar attributes will be rated similarly.

A Content-based Recommender System works with data that the user has provided, either explicitly or implicitly. Based on that data, a user profile is generated, and the recommendations are made by comparing the user profile with the content of each item in the database.

For this, it is necessary to assign an attribute or a vector with certain features that hold numeric or nominal values that represent the characteristics of each item. Only then it is possible to find similarities and dissimilarities between each element in the database and make an association with the user profile or items that the user liked in the past.

This means that is of high importance to have a well-represented set of features for the filtering technique to thrive.

As opposed to Collaborative Filtering, Content-based Filtering does not require any data regarding other users, since the recommendations are specific to each user, making it easier to scale to many users. It is also a good filtering technique in capturing specific interests of a user, allowing for the recommendation of niche items that most users are not interested in but that still hold very high interest for some. By collecting some initial inputs from users, user profiles begin to take shape and Content-based Filtering can make personalized recommendations early on, diminishing the impact of the cold-start problem that is so prevalent on the Collaborative Filtering technique.

However, considering that it is necessary to assign features to every item for a Content-based Filtering to work properly, meaning that it requires a lot of domain knowledge, this technique is only as good as the features assigned. Furthermore, being a technique that relies on the user profile and user past interest in other items, it has a very limited ability to expand the user's existing interests, leading to a lack of novelty and diversity in the recommendations provided. Although easily scalable for many users, the more items are added to the database, the more features need to be assigned, meaning that when presented with a large database of items, it does not scale well.

2.2.3 Knowledge-based Filtering

Knowledge-based Filtering differs from both Collaborative Filtering and Content-based Filtering in the sense that it does not require significant amounts of data from user's past interaction with the system. In fact, the principal cause for the cold-start problem mentioned in 2.1.1 and 2.2.2 is the lack of data (Bouraga, Jureta, Faulkner, & Herssens 2014), because in cases where the amount of information regarding users is limited, recommendations tend to be poor or lack full coverage over the entire spectrum of combinations for the user-item interactions, meaning that a knowledge-based Recommender System avoids this issue entirely.

This type of filtering technique relies on explicit requirements from the user, providing a more exploratory interaction with the options that are available in the system, and allowing a greater control over the recommendations that are made by the system.

The Knowledge-based filtering technique can usually be approached in two different methods:

- Constraint-based method

  This method is based on user specific requirements or constraints on the attributes of the items in the database. Domain-specific rules, that represent the knowledge the system has about the domain, are also used to match the constraints provided by the user. Furthermore, this method allows the creation of rules between the attributes of the user and the attributes of the items, assuming the user provides the required information, meaning that those rules may also be specified in the filtering process.

  In the event of a narrow list of results, the user has the possibility to update the initial constraints at any time in order to increase the number of results returned by the Recommender System.

- Case-based method

  In the Case-based method, similarity metrics are defined and assigned to the attributes of the items in the database so that when the user selects an example of the desired item, the system can retrieve similar items to the one chosen by the user, meaning that the similarity metrics must be carefully defined in a domain-specific way.

  The results retrieved by the Recommender System serve as new targets for the filtering process, along with slight modifications provided by the user, and each iteration brings the user towards the final recommendation.

Considering that this type of filtering relies on knowledge about the domain for which it is being implemented, it is not easily generalizable, albeit highly customized.

Knowledge-based Recommender Systems are often more reliable than Collaborative and Content-based Recommender Systems regarding recommendations and do not fall victim to the traditional cold-start problem. However, the construction of the knowledge base is usually a complicated task that demands considerable knowledge and expertise on specific domains.

2.2.4 Filtering Techniques Comparison

Each filtering technique has their own advantages and disadvantages and knowing which one to apply within a Recommender System is crucial to ensure the desired outcome. In Table 1 it is shown a comparison between the filtering techniques mentioned in the chapter 2.2.

Table 1 - Advantages and disadvantages of filtering techniques

	Advantages	Disadvantages
Collaborative Filtering	- Does not require extensive domain knowledge. - Quality of recommendations increase over time. - Allows for some novelty. - Recommendations are personalized.	- Does not scale very well. - There is usually a lot of sparsity in the data. - Does not handle new users or items very well.
Content-based Filtering	- Does not require information regarding users. - Easy to scale for a large user-base. - Captures specific interests of users. - Personalized recommendations require early on.	- Requires a lot of domain knowledge. - Leads to the lack of novelty and diversity. - Does not scale well for a large item-base.
Knowledge-based Filtering	- Often more reliable than Collaborative and Content-based Filtering. - Does not suffer from the Cold-Start problem. - Highly customizable.	- Demands considerable knowledge and expertise in the domain.

2.3 Related Work

Recommender Systems would most likely not come to be what they are today if not for the evolution that took place around 1970s, when the area of Recommender Systems turned as an independent area of research (Sharma & Singh 2016).

Since then, many studies have been conducted with the intention of better understanding the complexity around such systems and how to best apply them in the continuously growing diversity that the society offers.

Considering the nature of the Recommender System being developed for this dissertation, wide research on Recommender Systems related to Stress resulted in the selection of four articles:

1. Health Recommender Systems: Systematic Review
2. PopTherapy: Coping with Stress Through Pop-Culture
3. TeenRead: An Adolescents Reading Recommendation System Towards Online Bibliotherapy
4. mStress: A Mobile Recommender System for Just-In-Time Interventions for Stress

Although many documents were found related to the adoption of Recommender Systems in the scope of Health-related solutions, the lack of alternatives for Stress specific ones is notorious.

In fact, the first article was selected due to the wide documented review on Health Recommender Systems in general, and out of seventy-three published studies that were analyzed, only four were related to Recommender Systems in the scope of Stress.

2.3.1 Health Recommender Systems: Systematic Review

This article was created based on the premise that Health Recommender Systems offer the potential to motivate users to change their behavior by suggesting better choices and knowledge that users can act upon based on observed user behavior.

The main goal of this article was to make a wide review on the state of the art regarding Health Recommender Systems and how they apply to non-medical people.

The study was conducted according to the key steps required for systematic reviews according to PRISMA (Preferred Reporting Items for Systematic Reviews and Meta-Analyses) guidelines, resulting in the analysis of seventy three papers, and focused on literature related to Health Recommender Systems, including the type of recommendations made by the Recommender Systems, the recommender techniques that were utilized, how evaluations were approached, as well as how the recommendations were displayed to the users, and the transparency behind the rationale for the recommendations that were being given.

- Recommended Items

There is a wide variety of domains in which Health Recommender Systems operate and, in this article, four categories were identified regarding the recommendations: lifestyle (24 papers out of 73), nutrition (26 papers out of 73), general health information (23 papers out of 73), and specific health condition-related recommendations (9 papers out of 73). According to this article, a significant trend was found regarding the increase in popularity of nutrition-related advice.

- Recommender Techniques

Most of the Health Recommender Systems covered in this study rely on knowledge-based techniques (49 papers out of 73), either directly or within the context of a hybrid filtering approach, the reason being that this type of filtering is known for alleviating drawbacks of other filtering techniques such as cold-start and sparsity of data. Some use more straightforward-approaches such as if-else statements based on specific logic while others use more complex algorithms like particle swarm optimization, fuzzy logic, and reinforcement algorithms. Content-based techniques are also used (7 papers out of 73), as well as Collaborative (4 papers out of 73), although mostly in conjunction with other techniques, resulting in a hybrid approach.

- Evaluation Approach

This study categorizes the evaluations in two ways: offline evaluations (34 papers out of 73) and evaluations that rely on user interaction (39 papers out of 73).

For offline evaluations, the following metrics were reported: precision, accuracy, performance, recall, mean absolute error, normalized discounted cumulative gain, F1 score, and root mean square error.

For online evaluations, methods such as the use of surveys, single-session evaluations, in the wild, and randomized controlled tests were applied.

- Interface

More than half of the Health Recommender Systems covered in this study did not implement a graphical user interface to communicate the recommended health items to the user (39 papers out of 73), while the rest used either a mobile interface (18 papers out of 34) or web interface (14 papers out of 34) but not both.

- Transparency

Although this study highlights the importance of the reasoning behind the recommendations being made by the Health Recommender Systems, only a very small number (7 papers out of 73) of the covered systems do explain to the user why such recommendations are made.

The authors of this article state that there is a clear trend towards Health Recommender Systems that provide well-being recommendations but do not directly intervene in the user's medical status. It is also concluded that the most common filtering technique used in Health Recommender Systems is the hybrid approach, that evaluations of these systems vary greatly, and that although the presence of a user interface to display the recommendations, as well as the reasoning behind them, is very important, more than half of the covered studies lack this important feature.

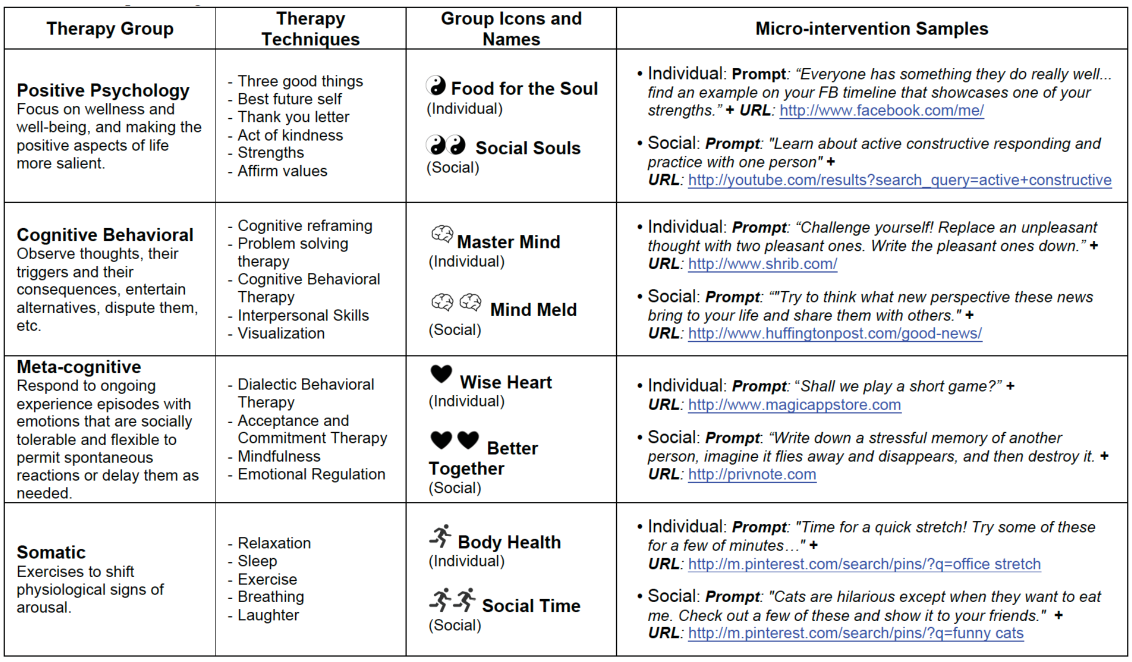
2.3.2 PopTherapy: Coping with Stress Through Pop-Culture

This article focuses on the potential of smart-phones as a conduit to provide stress-related therapy to the general population, considering how intrinsically present smart-phones are in our society. Recommendations are achieved through a mobile application and the experiment was conducted during four weeks for practical and monetary reasons.

Through the utilization of machine learning techniques, the goal of this article is to propose a solution that fits to as many people as possible by recommending interventions related to stress depending on users' personality and current needs.

The authors of this article designed and implemented a Windows Phone application and used cloud-based services to support the delivery of micro-interventions, collecting user feedback and contextual information.

By wanting to intersect stress management psychotherapy and popular web applications, the authors started by mapping the most used stress management psychotherapy approaches and eventually grouped them into four categories: positive psychology, cognitive behavioral, meta-cognitive, and somatic. Because socialization is an element associated with improved engagement, the authors further divided the four intervention groups into interventions that could be performed individually or collectively, as shown in Figure 2.

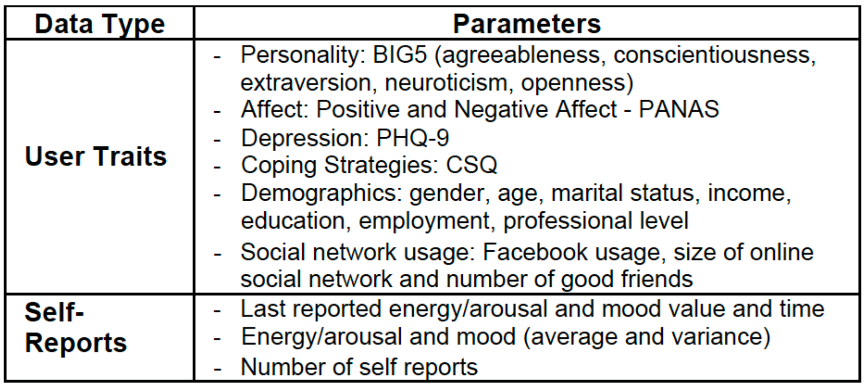


Therapy Group	Therapy Techniques	Group Icons and Names	Micro-intervention Samples
Positive Psychology Focus on wellness and well-being, and making the positive aspects of life more salient.	- Three good things - Best future self - Thank you letter - Act of kindness - Strengths - Affirm values	☯ **Food for the Soul** (Individual) ☯☯ **Social Souls** (Social)	• Individual: **Prompt**: *"Everyone has something they do really well... find an example on your FB timeline that showcases one of your strengths."* **+ URL**: http://www.facebook.com/me/ • Social: ***Prompt***: *"Learn about active constructive responding and practice with one person"* **+** ***URL***: http://youtube.com/results?search_query=active+constructive
Cognitive Behavioral Observe thoughts, their triggers and their consequences, entertain alternatives, dispute them, etc.	- Cognitive reframing - Problem solving therapy - Cognitive Behavioral Therapy - Interpersonal Skills - Visualization	🧠 **Master Mind** (Individual) 🧠🧠 **Mind Meld** (Social)	• Individual: ***Prompt***: *"Challenge yourself! Replace an unpleasant thought with two pleasant ones. Write the pleasant ones down."* **+** ***URL***: http://www.shrib.com/ • Social: ***Prompt***: *""Try to think what new perspective these news bring to your life and share them with others."* **+** ***URL***: http://www.huffingtonpost.com/good-news/
Meta-cognitive Respond to ongoing experience episodes with emotions that are socially tolerable and flexible to permit spontaneous reactions or delay them as needed.	- Dialectic Behavioral Therapy - Acceptance and Commitment Therapy - Mindfulness - Emotional Regulation	♥ **Wise Heart** (Individual) ♥♥ **Better Together** (Social)	• Individual: ***Prompt***: *"Shall we play a short game?"* **+** ***URL***: http://www.magicappstore.com • Social: ***Prompt***: *"Write down a stressful memory of another person, imagine it flies away and disappears, and then destroy it.* **+** ***URL***: http://privnote.com
Somatic Exercises to shift physiological signs of arousal.	- Relaxation - Sleep - Exercise - Breathing - Laughter	🏃 **Body Health** (Individual) 🏃🏃 **Social Time** (Social)	• Individual: ***Prompt***: *"Time for a quick stretch! Try some of these for a few of minutes…"* **+** ***URL***: http://m.pinterest.com/search/pins/?q=office stretch • Social: ***Prompt***: *"Cats are hilarious except when they want to eat me. Check out a few of these and show it to your friends."* **+** ***URL***: http://m.pinterest.com/search/pins/?q=funny cats

Figure 2 - Micro-intervention design matrix[2]

To match interventions to personal traits and temporal context, the authors required both input features and a machine learning model.

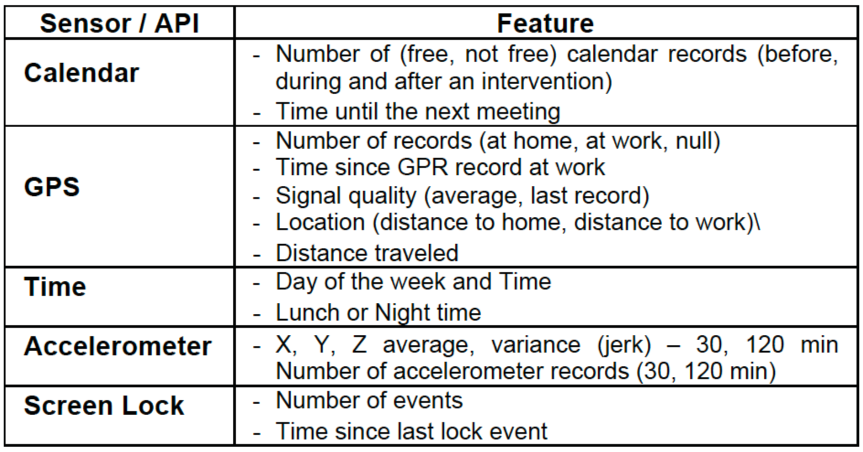
The input features came from both user and contextual data. The user data was obtained from a pre-study survey, but also by implementing an Experience Sampling Method where users were prompted to provide a self-assessment while using the mobile application. As for the contextual data, the authors used the phone sensors and APIs. The range of input features can be seen in Figure 3 and Figure 4.



Data Type	Parameters
User Traits	- Personality: BIG5 (agreeableness, conscientiousness, extraversion, neuroticism, openness) - Affect: Positive and Negative Affect - PANAS - Depression: PHQ-9 - Coping Strategies: CSQ - Demographics: gender, age, marital status, income, education, employment, professional level - Social network usage: Facebook usage, size of online social network and number of good friends
Self-Reports	- Last reported energy/arousal and mood value and time - Energy/arousal and mood (average and variance) - Number of self reports

Figure 3 - User data[2]

[2] Retrieved from (Paredes, Gilad-Bachrach, Czerwinski, & Roseway 2014)



Sensor / API	Feature
Calendar	- Number of (free, not free) calendar records (before, during and after an intervention) - Time until the next meeting
GPS	- Number of records (at home, at work, null) - Time since GPR record at work - Signal quality (average, last record) - Location (distance to home, distance to work)\\ - Distance traveled
Time	- Day of the week and Time - Lunch or Night time
Accelerometer	- X, Y, Z average, variance (jerk) – 30, 120 min Number of accelerometer records (30, 120 min)
Screen Lock	- Number of events - Time since last lock event

Figure 4 - Contextual data[3]

As for the machine learning model, the authors proposed modeling the problem as a contextual multi-armed bandit problem, meaning that they trained a model to predict the expected amount of stress reduced by each intervention for an individual at a given context. Based on the estimates, the system selects the intervention with the highest score while also exploring other interventions.

The machine learning model was trained using the Random Forest algorithm and the expected reduction of stress derived from the interventions was measured by calculating the delta between the subjective stress assessment before and after the interventions.

By using the Upper Confidence Bounds algorithms, the authors relied on optimistic predictions using the standard deviation computed from the deviation of the leaves of the trees that conform to the random forests of the average to find the intervention that is expected to reduce stress as much as possible and to tell which interventions to use in the future for the same purpose.

In this study, the authors decided to evaluate the impact of their approach on reducing stress through a mobile application rather than the performance of the recommender system that was created based on metrics such as descriptive statistics on interventions, stress deltas, and drop out ratios, and concluded that there is a potential for popular web applications to provide a source of stress management interventions. Furthermore, machine learning algorithms can be used to improve engagement and local efficacy by matching the right interventions to the context and personal traits of the user.

[3] Retrieved from (Paredes, Gilad-Bachrach, Czerwinski, & Roseway 2014)

2.3.3 TeenRead: An Adolescents Reading Recommendation System Towards Online Bibliotherapy

This article suggests online bibliotherapy as an effective way to diminish stress in adolescents through the recommendation of specific reading materials, avoiding the traditional burden of relying on someone with knowledge and expertise not only in psychology but also literature.

To overcome the fact that nowadays teenagers are most likely to read any kind of content through their mobile gadgets such as their mobile phones or laptops, the authors of this article developed a reading recommendation system called TeenRead, that is used as one of the components for a web application, with the purpose of recommending reading material to teenagers based on their stress and literature interests.

In this study it is stated that in bibliotherapy the content of the articles being recommended is very pertinent. For that reason, the authors opted for a content-based filtering technique and defined three guidelines which they followed to develop the recommender system: stress easing by reading, reading interests, and the unification of both.

For each user, the recommender system computes and maintains a stress easing effect vector, responsible for recording the stress easing effects by reading different categories of articles, and an interest vector, that records the interests of reading different categories and sub-categories of articles.

The user’s stress easing effect vector is defined as follows:

$$\begin{gathered}\bar{E} = (c1.ease, c2.ease, \cdots, c \mid SC \mid .ease), \\ \text{where for } \forall i \ (1 \leq i \leq \mid SC \mid) \wedge (ci \in SC) \wedge (ci.ease \in [0,5])\end{gathered} \tag{1}$$

SC is the set of articles’ categories (study, family, affection, inter-personnel, self-recognition, employment) and $|SC| = 6$.

The user’s interest vector is defined as follows:

$$\begin{gathered}\bar{I} = (cs1.int, cs2.int, \cdots, cs \mid ACs \mid .int), \\ \text{where for } \forall i \ (1 \leq i \leq \mid ACs \mid) \wedge (csi \in ACs) \wedge (csi.int \in [0,1])\end{gathered} \tag{2}$$

ACs is the set of all articles’ sub-categories and $|ACs|$ is the total number of articles’ sub-categories.

To measure a user’s interest in an article, the Root of Squares Sum of the interest values of the article’s sub-categories is calculated through the following function, where $ACs(article)$ returns the set of sub-categories that the article falls into:

(3)

$$Interest(\bar{I}, article) = \sqrt{\sum_{cs \in ACs(article)} cs.int^2}$$

In the same way, to measure an article's easing effect, the Root of Squares Sum of the article's categories' easing effects is calculated through the following function, where $SC(article)$ returns the set of categories that the article falls into:

$$Ease(\bar{E}, article) = \sqrt{\sum_{c \in SC(article)} c.ease^2} \tag{4}$$

Considering the equations for Interest and Ease, the recommendation score of an article to a user can be defined as follows:

$$Recommend(\bar{E}, \bar{I}, article) = \rho * Ease(\bar{E}, article) + (1 - \rho) * Interest(\bar{I}, article), \quad \text{where } \rho \in [0,1] \tag{5}$$

ρ is a coefficient for adjusting the weights of the stress easing effect and reading interest.

In the beginning, when $\bar{E} = (0,\cdots,0)$ and $\bar{I} = (0,\cdots,0)$, the recommendation scores will be 0, meaning that the recommender system will randomly select articles to recommend. However, as both the stress easing effect and interest vectors get updated over time, through the usage of a content-based filtering, a similarity function is used to calculate similarities between sub-categories and therefore recommend different articles to the user.

The authors of this study state that TeenRead was in internal test at the time of writing and therefore was subjected only to ten volunteers. To test the efficacy of the recommender system, eleven reading themes were prepared from which each volunteer had to select six that they had the most interest for. The recommendation was updated every seven days, meaning that the volunteers had to switch themes every seven days. By the end of those seven days, the volunteers had to report about the stress easing effects and the interest on the recommended articles. Through study statistics that are not reported in this study, the authors state that TeenRead demonstrated good effectiveness by decreasing the stress level of users up to 22%.

2.3.4 mStress: A Mobile Recommender System for Just-In-Time Interventions for Stress

This article also focuses on the presence of smart-phones on our day-to-day lives, but also on other smart devices such as smart-watches and smart-bands, to build a mobile application capable of detecting levels of stress, through the integration with a smart-band, and recommend stress-related interventions accordingly.

The mobile recommender system is composed of three main modules, namely the stress detection module, the intervention module, and the recommendation module.

- Stress detection module

Considering that there are studies that show that people with high levels of work stress have higher heart rates during work hours and leisure time after work, the authors relied on the heart rate measurements from a Microsoft Band 2 to detect if the user is under stress. Using a device as small as a smart band, the goal is to detect stress mostly unobtrusively and for long periods of time.

One hundred heart beats per minute was the threshold defined by the authors to tell if the user is under an episode of stress, as long as the heart beats keep going at or over that threshold for more than a minute, avoiding taking into consideration situations of sudden movement or excitement. Because the Microsoft Band 2 can recognize various types of motions from the user, stress levels under physical activity are disregarded by the recommender system.

Assuming the user is under significant levels of stress, an intervention from the recommender system is sent to the user after a small delay.

- Intervention module

The interventions that were chosen by the authors, just like in 2.4.2, fall into four categories of psychotherapy approaches: positive psychology, cognitive behavioral, meta-cognitive, and somatic.

A total of eighteen interventions were created and are done through separate mobile applications from the Google Play store. These applications had to be free and engaging for the user, so the authors relied on the ratings in the Google Play store when choosing them.

We can see four examples of just-in-time interventions used by the intervention module in Figure 5.

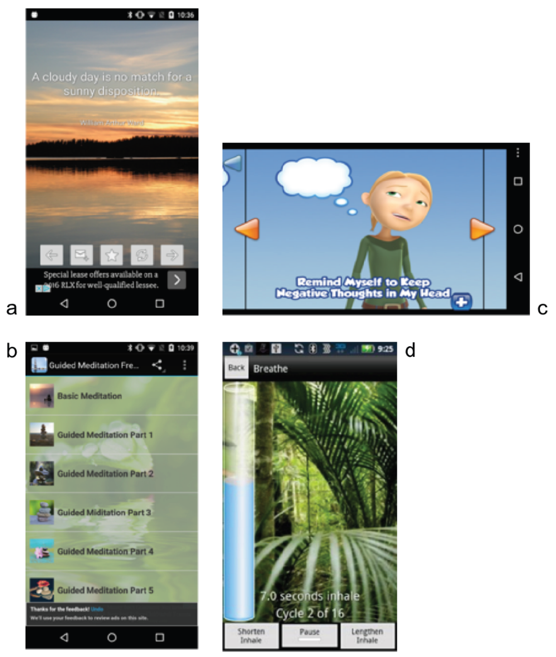


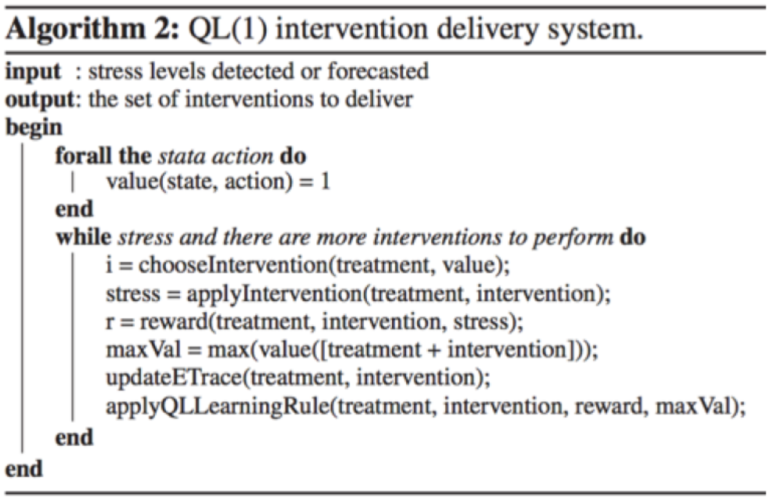
Figure 5 - Four examples of psychotherapy approaches[4]

- Recommendation module

The recommendation module is based on a learning model for intervention delivery and uses a Q-learning algorithm with legibility traces to select the combination of interventions that relieves the user from stress while intervening the least number of times possible. For every state in Q-learning, there is an expected reward in the future based on every action that is taken.

Figure 6 shows the algorithm upon which the construction of the recommendation module described in this article was based on.

[4] Retrieved from (Clarke, Jaimes, & Labrador 2017)



```
Algorithm 2: QL(1) intervention delivery system.
input  : stress levels detected or forecasted
output: the set of interventions to deliver
begin
    forall the stata action do
        value(state, action) = 1
    end
    while stress and there are more interventions to perform do
        i = chooseIntervention(treatment, value);
        stress = applyIntervention(treatment, intervention);
        r = reward(treatment, intervention, stress);
        maxVal = max(value([treatment + intervention]));
        updateETrace(treatment, intervention);
        applyQLLearningRule(treatment, intervention, reward, maxVal);
    end
end
```

Figure 6 - Q-learning algorithm basis[5]

The recommendation module starts by choosing interventions at random, learning how each intervention affects the user by receiving a reward value. To calculate the expected reward, or Q value, authors used the following equation:

$$Q(state, action) = R(state, action) + \gamma * Max[Q(next\ state, all\ actions)] \quad (6)$$

The constant γ can assume any value between 0 and 1 and the closer this constant is to 0, the more the system will consider the immediate reward over the Q-value when choosing between interventions.

For each intervention, depending on whether stress was relieved, diminished, or remained the same, the reward would vary from 100, to 50, to 0, respectively, resulting in the implementation of the algorithm shown in Figure 7.

[5] Retrieved from (Clarke, Jaimes, & Labrador 2017)

```
Algorithm 1: Intervention Recommender System.
input: current state
output: intervention to be given
while there is stress do
    for all actions do
        $Q(state, action) = R(state, action) +$
        $\gamma * Max[Q(next\ state, all\ actions)]$
    end
    apply intervention;
    if stress was relieved then
        R(state,action) = 100;
    end
    if stress decreased but not relieved then
        R(state, action) = 50;
    else
        R(state, action) = 0;
    end
end
```

Figure 7 - Q-learning algorithm implementation[6]

For testing purposes, the authors of this article limited their evaluations to the execution of one thousand episodes of stress, concluding that the optimal value for γ is 0,8. Although Q-learning required more than six interventions to effectively relief stress in the beginning, by the time Q-learning reached its one hundredth execution, the average number of interventions reduced to less than two consistently, eventually reaching its minimum average of 1,1 interventions necessary to relief stress by the end of the execution of all the stress episodes.

[6] Retrieved from (Clarke, Jaimes, & Labrador 2017)

3 Value Analysis

In this chapter the value analysis is presented regarding the purpose of this project as a Recommender System.

This chapter begins with the identification and analysis of the opportunity, followed by the value proposition according to the Osterwalder model, and concludes with an evaluation and selection of the filtering technique that should be used for the development of the Recommender System.

3.1 Opportunity

Humans are the key to any successful business venture. It is in human nature to constantly thrive for more and achieve the next goal, but success often comes at a great price. More than ever, people define their selves based on their job. This may be a whole other problem but is tied to the feeling of responsibility that people undergo through during their day-to-day job.

Demanding markets, fast paced innovation, and the fear of missing out, are some of the motives that lead to long working hours, difficult tasks to overcome, ever-increasing backlog, and increased workload.

Because workers feel responsible for the resolution of the constant challenges they regularly face at their job, the idea that they also must perform at their best and at a constant pace, at the fear of losing their job, often leads to emotional or physical distress. This can be manifested as stress and has a negative impact not only on workers but on the entities that employ them.

There are ways to mitigate or even avoid the side effects of stress, but it is difficult to apply such techniques at a major scale and in a systematic way. Hence, the responsibility to deal with it is often relayed to the workers, sometimes through optional initiatives from the companies they work for, but mostly through professional advice.

As technology advances, it becomes possible to bring products and services directly to people for different purposes. Therefore, considering the impact and consequences of occupational stress, companies begin to proactively seek ways to overcome its effects.

Stress manifests differently in every individual, the causes may not be the same for everyone, and there is a wide range of approaches that can be taken to address it. There are people who specialize in the treatment and avoidance of mental health disorders such as stress, whether through medical advice, specific physical and mental activity, or adoption of healthy regular habits.

Considering the continuously increasing digitalization, the development of an application capable of detecting stress, through the capture of images, and recommending actions upon the levels of stress being detected, may provide a way not only for companies to control and eventually prevent episodes of stress of their workers during their day-to-day jobs, but also for workers to become more self-aware regarding the negative effects of uncontrolled stress and how to alleviate them.

In Table 2, SWOT analysis is presented regarding the implementation of a solution for detection of stress and recommendation of stress-relieving actions.

Table 2 - SWOT analysis

Strengths	- Stress monitoring. - Personalized recommendations. - Self-awareness. - Avoidance of stress related issues. - More stable workforce.
Weaknesses	- Reliance on desktop platform. - Difficult to measure short-term impact.
Opportunities	- Support stress detection through other means such as smart bands. - Development of better filtering techniques.
Threats	- Lack of motivation to use the application. - Unwillingness to provide ratings to recommendations. - Lack of vision regarding the impact that stress has.

3.2 Value Proposition

The value proposition for this solution is presented according to the Osterwalder model, composed of two main blocks – customer profile and the value proposition of the company. In this case, the solution to be implemented shall be used by workers while on their day-to-day tasks at their jobs, hoping to grow self-awareness related to stress and how to behave according to the levels of stress detected, contributing for a healthier lifestyle, and preventing more serious health issues.

In the representation of the Osterwalder model in Figure 7, for the customer segment it is shown the following:

- Gains: The benefits desired and wished by the customer.
- Pains: The negative experiences associated with the present goal.
- Customer Jobs: The problems being addressed.

Regarding the value proposition of the company, also in Figure 8, it is shown the following:

- Gain Creators: How the solution being implemented adds value to the customer.
- Pain Relievers: How the solution being implemented relieves customer pains.
- Products and Services: Which products and services are expected to address the previous points.

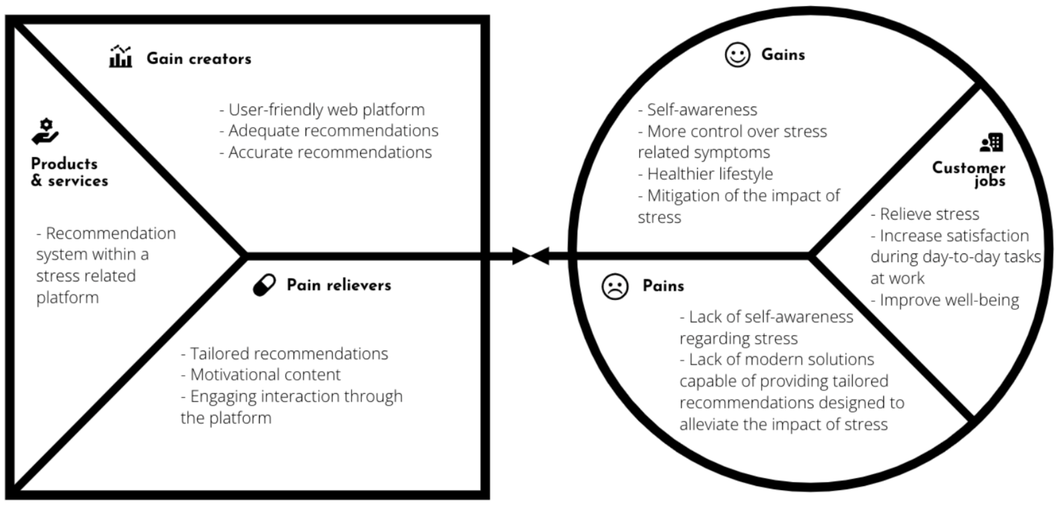

Gain creators
- User-friendly web platform
- Adequate recommendations
- Accurate recommendations
Products & services
- Recommendation system within a stress related platform
Pain relievers
- Tailored recommendations
- Motivational content
- Engaging interaction through the platform
Gains
- Self-awareness
- More control over stress related symptoms
- Healthier lifestyle
- Mitigation of the impact of stress
Customer jobs
- Relieve stress
- Increase satisfaction during day-to-day tasks at work
- Improve well-being
Pains
- Lack of self-awareness regarding stress
- Lack of modern solutions capable of providing tailored recommendations designed to alleviate the impact of stress

Figure 8 - Value proposition based on Osterwalder model

Considering the intent of the solution being described in this document, the customer segment addressed consists of everyone that identifies as a worker within a company that may rely not only, but also on the interaction with a computer with access to the internet and a camera so that images are captured, analyzed, and recommendations generated according to the levels of stress detected.

3.3 Functional Analysis

Functional Analysis and System Technique (FAST) is used to define, analyze, and understand the functionalities of a solution, as well as how those functionalities correlate between each other. It is represented through a diagram to better provide a visualization of the functionalities in a logical sequence.

The diagram can be interpreted not only from left to right, beginning with the core functionality of the solution and further exploring the functionality to a more granular level, but also from right to left, indirectly explaining the reason behind each functionality.

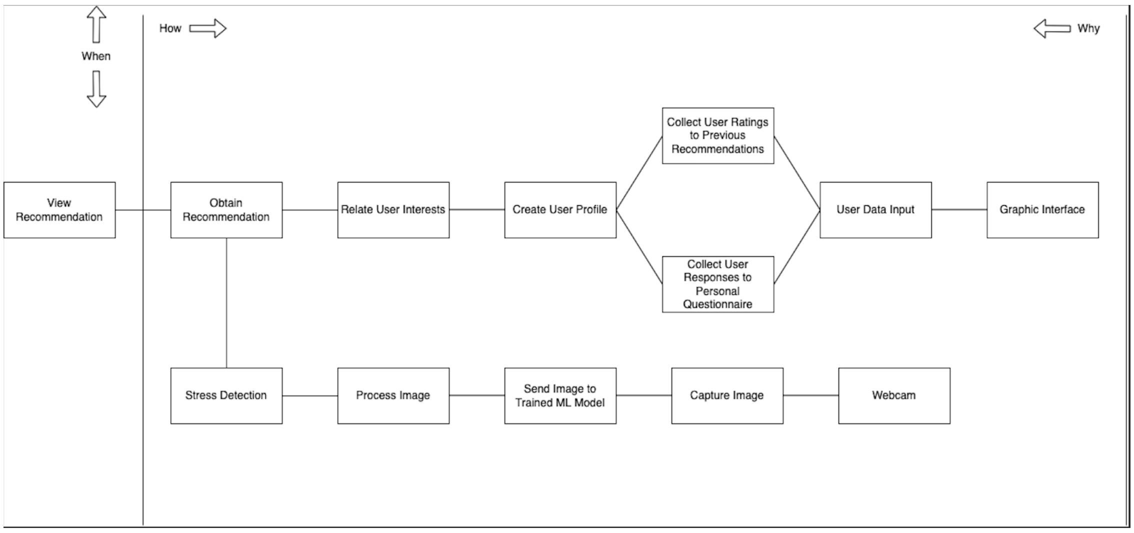


Figure 9 - FAST diagram

In Figure 9 it is possible to identify the following functionalities:

- Obtain recommendation
- Correlate user interests and preferences
- Create user profiles
- Collect user answers to questionnaires
- Collect user ratings regarding past recommendations
- Take user input
- Process images
- Capture images

The main purpose of the solution is to provide stress-easing recommendations. However, recommendations should be provided at the right time, meaning that the solution must be capable of detecting signs of stress. This shall be achieved through the capture of images through the webcam of a computer, that is then sent to a trained machine learning model. Meanwhile, the main objective is to obtain those recommendations. Through the graphical user interface, users can provide their input through questionnaires and ratings of previous recommendations, resulting in the creation of a user-profile. Through this profile, it shall be possible to provide the desired recommendations using a filtering technique and a database of proper stress-easing actions.

3.4 Filtering Technique Selection

Selecting the right filtering technique for a recommender system is crucial, for filtering techniques are the backbone of the logic behind recommendations being made. Thus, *Analytic Hierarchy Process (AHP)* is used to analyze and aid in the selection of the filtering technique to be used.

This process is composed of seven phases:

1. Construction of the decision hierarchic tree
2. Comparison between the elements of the hierarchy
3. Relative priority of each criterion
4. Evaluation of consistency between relative priorities
5. Construction of the comparison matrix for each criterion
6. Calculation of the composite priority for the alternatives
7. Selection of the alternative

For the alternatives, the most common filtering techniques were selected:

- Collaborative Filtering
- Content-based Filtering
- Knowledge-based Filtering

The alternatives are compared between themselves through the following criteria:

- User scalability
- Item scalability
- Domain knowledge

In Figure 10, it is possible to see the previous described elements in a decision hierarchic tree.

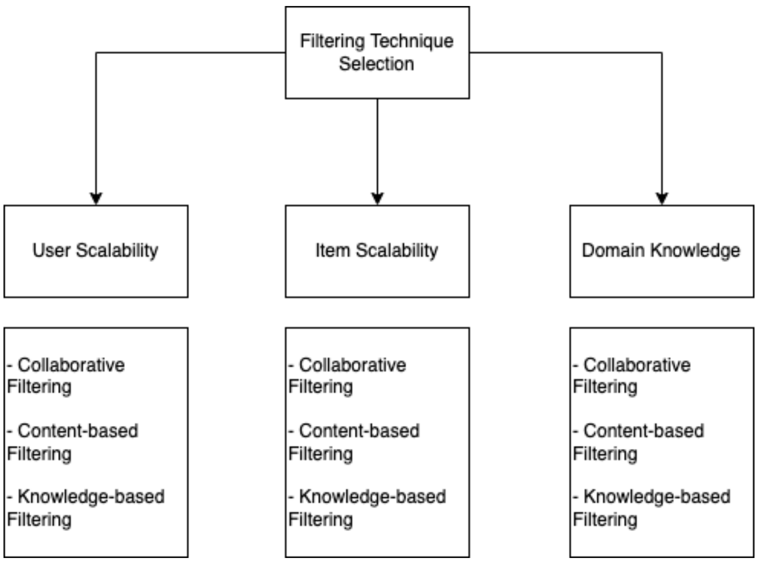
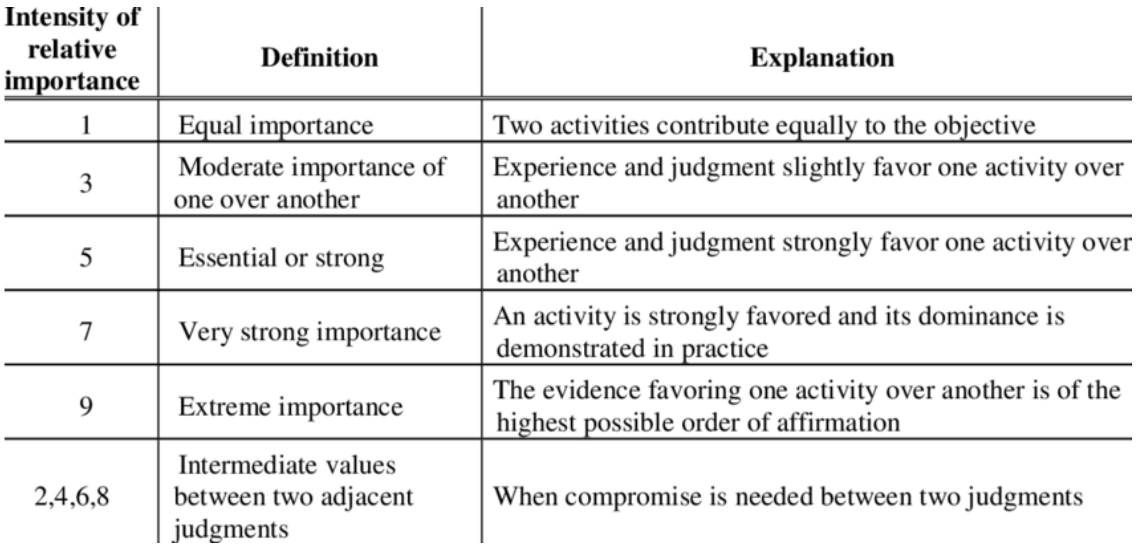

Filtering Technique Selection
User Scalability
Item Scalability
Domain Knowledge
- Collaborative Filtering
- Content-based Filtering
- Knowledge-based Filtering
- Collaborative Filtering
- Content-based Filtering
- Knowledge-based Filtering
- Collaborative Filtering
- Content-based Filtering
- Knowledge-based Filtering

Figure 10 - Decision Hierarchic Tree

Considering the decision hierarchic tree in Figure 10, it is necessary to define the level of importance of each criterion. For this, the fundamental scale of Saaty is used, represented in Figure 11.



Intensity of relative importance	Definition	Explanation
1	Equal importance	Two activities contribute equally to the objective
3	Moderate importance of one over another	Experience and judgment slightly favor one activity over another
5	Essential or strong	Experience and judgment strongly favor one activity over another
7	Very strong importance	An activity is strongly favored and its dominance is demonstrated in practice
9	Extreme importance	The evidence favoring one activity over another is of the highest possible order of affirmation
2,4,6,8	Intermediate values between two adjacent judgments	When compromise is needed between two judgments

Figure 11 - Fundamental scale of Saaty[7]

Mapping the fundamental scale of Saaty to the decision hierarchic tree in Figure 10, the following level of priorities for each criterion can be represented through Table 3.

[7] A methodology for improving productivity of the existing shipbuilding process using modern production concepts and the AHP method 2017, Stanic, V., & Fafandjel, N., https://doi.org/10.21278/brod68303

Table 3 - Criteria comparison matrix

	User Scalability	Item Scalability	Domain Knowledge
User Scalability	1	1/5	1/2
Item Scalability	5	1	3
Domain Knowledge	2	1/3	1

Then, the criteria comparison matrix is normalized to obtain the relative priority of each criterion. For this, it is necessary to divide each value by its columns sum, allowing for the calculation of the relative priority, used in the priority vector, and it can be represented through Table 4.

Table 4 - Normalized criteria comparison matrix

	User Scalability	Item Scalability	Domain Knowledge	Relative Priority
User Scalability	1/8	3/23	1/9	0,12
Item Scalability	5/8	15/23	6/9	0,65
Domain Knowledge	2/8	15/69	2/9	0,23

The next step consists of calculating the Consistency Ratio (CR) to measure how consistent the judgments have been. If the CR is greater than 0.1, it means the judgements are untrustworthy because they are too close to randomness and the process should be reanalyzed and repeated.

To determine the value of CR, it is necessary to calculate the Consistency Index (CI), as well as the Random Index (RI).

The Consistency Index can be calculated as such:

$$CI = \frac{\lambda_{max} - n}{n - 1} \tag{7}$$

Where λ_{max} corresponds to the greatest value of the comparison matrix and n to the order of the matrix.

The Random Index, however, refers to many pairwise comparisons made and was calculated for square matrices of order n by the Oak Ridge National Laboratory, in the United States, resulting in the Table 5, where the variance of RI depends on the number of criteria.

Table 5 - Values of RI for square matrices of order n

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
RI	0.00	0.00	0.58	0.90	1.12	1.24	1.32	1.41	1.45	1.49	1.51	1.48	1.56	1.57	1.59

Considering the number of criteria taken into consideration for this case, the value of RI is 0,58.

Therefore, it is possible to calculate the Consistency Ratio through the following formula:

$$CR = \frac{CI}{RI} \tag{8}$$

Taking the formula for the calculation of the CI, it is first necessary to determine the value of λ_{max}:

$$\begin{bmatrix} 1 & 1/5 & 1/2 \\ 5 & 1 & 3 \\ 2 & 1/3 & 1 \end{bmatrix} \begin{bmatrix} 0,12 \\ 0,65 \\ 0,23 \end{bmatrix} = \lambda_{max} \begin{bmatrix} 0,12 \\ 0,65 \\ 0,23 \end{bmatrix} \Leftrightarrow$$

$$\Leftrightarrow \begin{bmatrix} 0,37 \\ 1,94 \\ 0,69 \end{bmatrix} = \lambda_{max} \begin{bmatrix} 0,12 \\ 0,65 \\ 0,23 \end{bmatrix} \Leftrightarrow \tag{9}$$

$$\Leftrightarrow \lambda_{max} = \begin{bmatrix} 3,08 \\ 2,98 \\ 3 \end{bmatrix}$$

To obtain the value corresponding to the matrix of λ_{max}, the average is calculated as follows:

$$\lambda_{max} = \frac{3,08 + 2,98 + 3}{3} \cong 3,02 \tag{10}$$

Then, it is possible to obtain the value of the Consistency Index as such:

$$CI = \frac{3,02 - 3}{3 - 1} \cong 0,01 \tag{11}$$

Considering the following CI with a value of 0,01 and the RI with a value of 0,58, CR can be calculated as follows:

$$CR = \frac{0,01}{0,58} \cong 0,017 \tag{12}$$

Having a Consistency Ratio lesser than 0,1, it is possible to conclude that the judgements are consistent, which allows for this process to continue to the next step of creating the comparison matrices by criteria.

The first matrix corresponds to User Scalability. The evaluation of this criterion is based on how well each alternative adapts to the user-base growth.

Table 6 - User Scalability comparison matrix

User Scalability	Collaborative Filtering	Content-based Filtering	Knowledge-based Filtering
Collaborative Filtering	1	1/9	1/9
Content-based Filtering	9	1	1
Knowledge-based Filtering	9	1	1

Table 7 - Normalized User Scalability comparison matrix

User Scalability	Collaborative Filtering	Content-based Filtering	Knowledge-based Filtering	Priority Vector
Collaborative Filtering	1/19	1/19	1/19	0,052
Content-based Filtering	9/19	9/19	9/19	0,474
Knowledge-based Filtering	9/19	9/19	9/19	0,474

The second matrix corresponds to Item Scalability. The evaluation of this criterion is based on how well each alternative adapts to the item-base growth.

Table 8 - Item Scalability comparison matrix

Item Scalability	Collaborative Filtering	Content-based Filtering	Knowledge-based Filtering
Collaborative Filtering	1	1/3	1/3
Content-based Filtering	3	1	7
Knowledge-based Filtering	3	1/7	1

Table 9 - Normalized Item Scalability comparison matrix

Item Scalability	Collaborative Filtering	Content-based Filtering	Knowledge-based Filtering	Priority Vector
Collaborative Filtering	1/7	21/93	1/25	0,14
Content-based Filtering	3/7	21/31	21/25	0,65
Knowledge-based Filtering	3/7	21/217	3/25	0,21

The third matrix corresponds to Domain Knowledge. The evaluation of this criterion is based on how independent a filtering technique is to the knowledge regarding the domain in order to provide proper recommendations.

Table 10 - Domain Knowledge comparison matrix

Domain Knowledge	Collaborative Filtering	Content-based Filtering	Knowledge-based Filtering
Collaborative Filtering	1	1/3	1/3
Content-based Filtering	3	1	7
Knowledge-based Filtering	3	1/7	1

Table 11 - Normalized Domain Knowledge comparison matrix

Domain Knowledge	Collaborative Filtering	Content-based Filtering	Knowledge-based Filtering	Priority Vector
Collaborative Filtering	1/7	21/93	1/25	0,14
Content-based Filtering	3/7	21/31	21/25	0,65
Knowledge-based Filtering	3/7	21/217	3/25	0,21

The results from both the relative priority matrix of each criterion and the matrices of the priority vector of each alternative can better be represented in Figure 12.

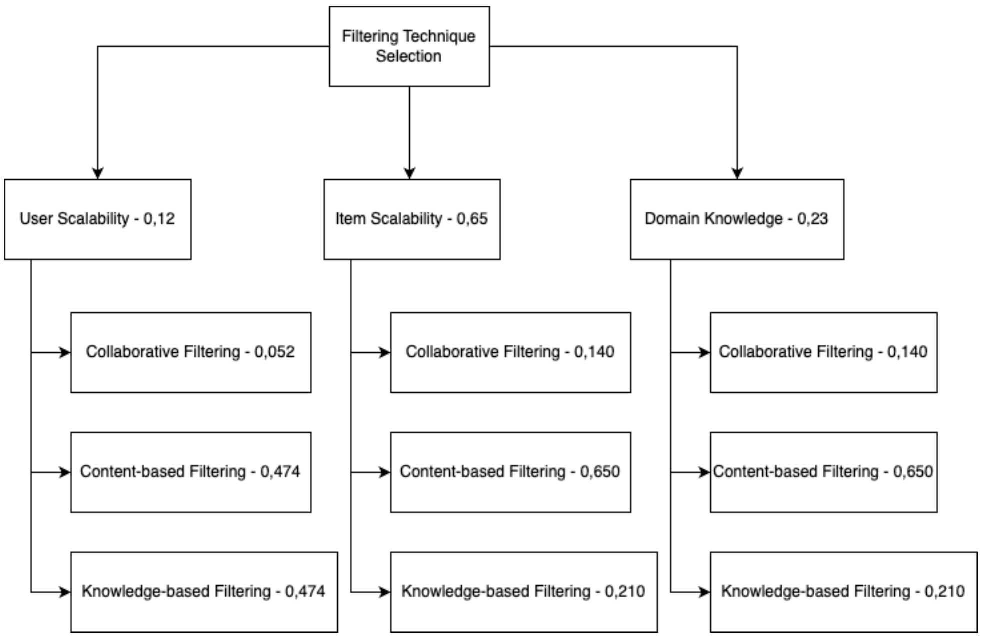

Filtering Technique Selection
User Scalability - 0,12
Item Scalability - 0,65
Domain Knowledge - 0,23
Collaborative Filtering - 0,052
Content-based Filtering - 0,474
Knowledge-based Filtering - 0,474
Collaborative Filtering - 0,140
Content-based Filtering - 0,650
Knowledge-based Filtering - 0,210
Collaborative Filtering - 0,140
Content-based Filtering - 0,650
Knowledge-based Filtering - 0,210

Figure 12 - Weighted Decision Hierarchic Tree

Lastly, it is possible to select the alternative that is best suited by multiplying the relative priority matrix of each criterion with the matrix of the priority vector of each alternative, as such:

$$\begin{bmatrix} 0{,}052 & 0{,}140 & 0{,}140 \\ 0{,}474 & 0{,}650 & 0{,}650 \\ 0{,}474 & 0{,}210 & 0{,}210 \end{bmatrix} \begin{bmatrix} 0{,}12 \\ 0{,}65 \\ 0{,}23 \end{bmatrix} = \begin{bmatrix} 0{,}127 \\ 0{,}629 \\ 0{,}056 \end{bmatrix} \tag{13}$$

Considering the criteria that were defined, as well as their relative significance, and according to these results, it is possible to conclude that the second row scored a greater value than the other two, having a value of 0,629, meaning that the filtering technique that should be chosen is the Content-based Filtering.

4 Analysis and Design

This chapter describes the steps involved in the analysis and design for this project. As result from the several meetings with the stakeholders, both functional and non-functional requirements are described, as well as the domain model and use cases proposed for this solution. The architecture from a component perspective is presented, where each component present in the architecture is described in detail. Furthermore, two architectures are proposed, explaining the reason behind the selection of one over the other.

4.1 Functional Requirements

Functional requirements aim to describe the principal functionalities of a system regarding the business and domain for which it is proposed, and can be described as following:

1. The system shall provide a questionnaire for the signed in user to reply to.
2. The system shall allow users to update past answers to the questionnaire.
3. The system shall display notifications for the latest recommendations.
4. The system shall allow the rating of past recommendations.
5. The system shall display the history of past recommendations.
6. The system shall provide recommendations.

4.2 Non-Functional Requirements

Unlike functional requirements, instead of what the system should do, non-functional requirements focus on how the system should be developed and the associated constraints.

To describe the non-functional requirements, it is utilized the FURPS+ model, for this model allows the classification of software quality attributes in a standard and more granular manner. The acronym stands for Functionality, Usability, Reliability, Performance, and Supportability. As this model evolved, the plus sign was added to further define constraints in design, implementation, interface, and hardware (Chung, Nixon, Yu, & Mylopoulos 2000).

In the context of non-functional requirements, Functionality represents the requirements that are not part of the use cases but still relevant to the solution domain-wise, and can be described as following:

1. The system shall be capable of generating personalized recommendations.
2. The system shall provide an authentication and authorization mechanism.
3. The system shall forbid access to unauthenticated and unauthorized users.
4. The system shall save user answers related to the questionnaire.

5. The system shall save user ratings related to the recommendations.

Usability refers to graphical interface constraints, such as accessibility, aesthetics, and consistency. The following requirements fall under this category:

6. The graphical interface shall be user-friendly, interactive, and engaging.
7. The graphical interface shall follow a pre-defined set of colors and components.

Reliability refers to the integrity and interoperability of the software, focusing on failure frequency and severity, mean time between failures, and data recovery. No requirements were identified that fall into this category.

Performance consists in the measurement of the software solution regarding response time, memory consumption, and processing power. The following requirement falls under this category:

8. The system shall be scalable.

Supportability refers to the concerns regarding maintenance, compatibility, and extensibility of the software. The following requirements fall under this category:

9. The graphical interface shall be supported by multiple web browsers such as Chrome, Firefox, Edge, and Opera.
10. The system shall be designed so that future developments are not compromised.

Design constraints focus on the limitations of the solution regarding tools and technologies. The following requirement falls under this category:

11. All data shall be stored in a relational database.

Interface constraints refer to the specifications regarding external and internal interactions between various components that make part of the software solution. The following requirements fall under this category:

12. The communication between components shall be done through REST (Representational State Transfer) when applicable.
13. The communication between components shall be done through a Message Broker, for non-blocking executions, when applicable.
14. Notifications from the server to the client shall not rely on RESTful communication.

Regarding implementation and hardware constraints, no requirements were identified that fall into these categories.

4.3 Use Cases

Considering the requirements specified in subchapter 4.2, it is possible to represent the interaction of a user with the system through a use cases diagram, as shown in Figure 13:

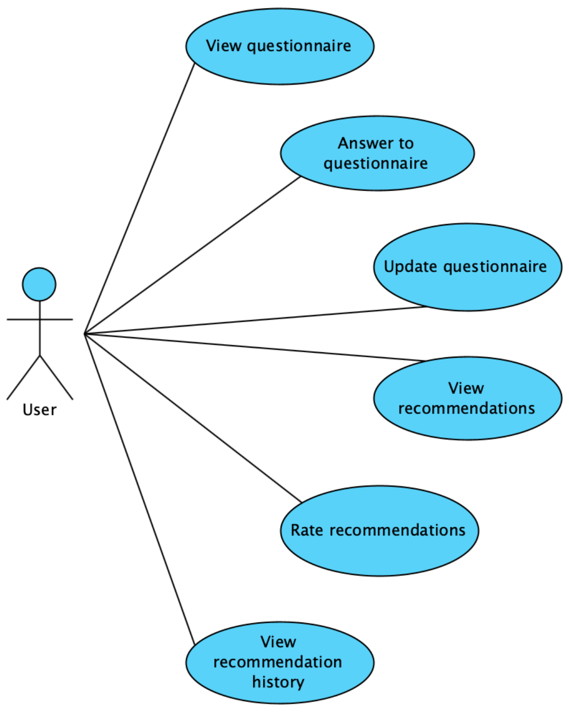


Figure 13 - Use Cases Diagram

The use cases represented in Figure 13 can be described as following:

- **View questionnaire**: The user navigates to the questionnaire page and is able to read the displayed questions.
- **Answer to questionnaire**: The user navigates to the questionnaire page and is able to select an answer from the list of possible answers for each question, or answer freely if no options are provided.
- **Update questionnaire**: The user navigates to the questionnaire page and is able to change a past answer to any of the displayed questions.
- **View recommendations**: The user receives a recommendation through interface notifications and is able to click on it to further expand the recommendation details.
- **Rate recommendations**: The user is able to classify the satisfaction towards a recommendation through the interface, when further expanding the recommendation details, on a scale from one to five.

- **View recommendation history**: The user navigates to the recommendation history page and is able to interact with past recommendations.

4.4 Domain Model

The domain model can be described as a detailed presentation of the real-world entities and their relationships within the scope of a business domain of a solution. Having system requirements in mind, identifying domain entities and relationships provide a solid basis that promotes the creation of systems for maintainability and incremental development. Furthermore, domain modeling helps reducing redundancy and improving integrity of the solution (Knaster 2021).

In Figure 14, it is shown the domain model proposed for this solution:

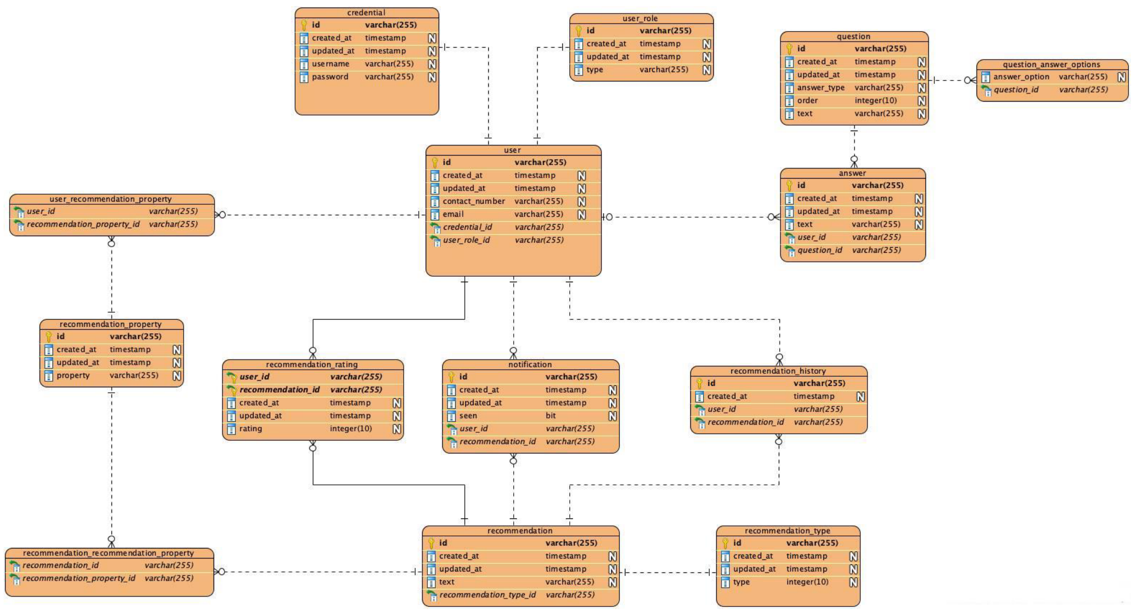


Figure 14 - Domain Model

The identified entities and their corresponding relationships can be described as follows:

- **user**: Represents the entity that interacts with the system.
- **credential**: Represents the authentication credentials belonging to the user.
- **user_role**: Represents the roles the user may be assigned to.
- **question**: Represents the questions in the questionnaire that the user shall reply to.
- **question_answer_options**: Represents the possible answers to the many questions in the questionnaire.
- **answer**: Represents the answers from the user to the questions in the questionnaire.
- **recommendation**: Represents the information that is recommended to the user.
- **recommendation_type**: Represents the type of information that is recommended to the user.

- **recommendation_rating**: Represents the rating that the user may assign to the information that is recommended.
- **recommendation_history**: Represents the information that was recommended to the user in the past.
- **recommendation_property**: Represents the properties of the information that is recommended to the user.
- **recommendation_recommendation_property**: Represents the many-to-many relationship between entities recommendation and recommendation_property, for a recommendation may have multiple recommendation properties and the same recommendation property may be assigned to multiple recommendations.
- **user_recommendation_property**: Represents the many-to-many relationship between entities user and recommendation_property, for a user may have multiple recommendation properties and the same recommendation property may be assigned to multiple users.
- **notification**: Represents the messages that are sent from the system to the user.

4.5 Architecture

Software development varies greatly, as does the architecture on which software solutions are based on. There is almost always more than a single path to achieve any goal, hence two alternatives are presented through a components diagram, as well as which alternative was chosen and the reason for that decision.

The traditional approach for the architecture of a software solution is known as the Monolithic Architecture. This type of architecture has brought success for many software solutions due to the ease of development, testing and deployment, that is associated with the fact that every element within the solution is under the same component.

Some advantages of a monolithic approach are:

- Simplicity of development
- Simplicity of testing
- Simplicity of deployment
- Customizations are used only one
- Simplicity on the onboarding of new members to a team

However, the monolithic approach comes with some disadvantages, such as:

- Lack of flexibility
- High coupling
- Lack of ownership within the solution
- Scaling issues

In Figure 15, it is presented the architectural proposal for this project in a monolithic approach.

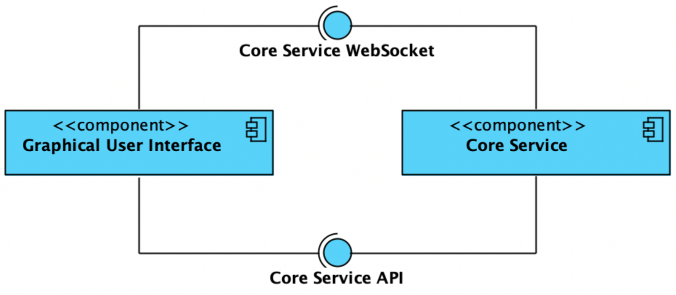


Figure 15 - Monolithic oriented architecture

In the fast-paced, fast-growing business environment, the short-comings of the monolithic approach became apparent, which led to the adoption of another type of architecture for software solutions. This type of architecture is known as the Micro-services Architecture, that encapsulates each business capability into individual smaller pieces of software within the solution.

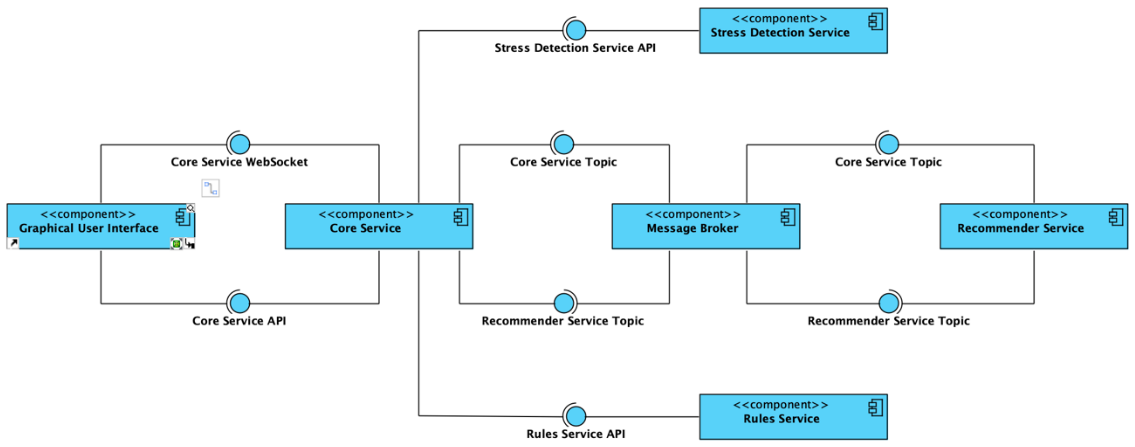
Some advantages of a micro-services approach are:

- Improved fault isolation
- Low coupling
- Smaller and faster deployment
- Highly scalable
- Clear domains
- Highly flexible

The drawbacks of choosing a micro-services approach are:

- Increased complexity in service communication
- Additional monitoring
- Increased complexity in solution-wide testing
- Increased demand of resources
- Increased complexity in deployment

In Figure 16, it is presented the architectural proposal for this project in a micro-services approach.



<<component>> Stress Detection Service
Stress Detection Service API
Core Service WebSocket
Core Service Topic
Core Service Topic
<<component>> Graphical User Interface
<<component>> Core Service
<<component>> Message Broker
<<component>> Recommender Service
Core Service API
Recommender Service Topic
Recommender Service Topic
<<component>> Rules Service
Rules Service API

Figure 16 - Micro-services oriented architecture

Considering the necessities of this project and comparing the advantages and disadvantages of one architectural approach to another, the micro-services architecture is considered the best alternative, mainly due to its scalability and flexibility.

Both approaches are composed of the same business units:

- Graphical User Interface
- Core Service
- Rules Service
- Recommender Service
- Stress Detection Service
- Message Broker

The Graphical User Interface shall be created using a JavaScript library called React. This library is open-source and is maintained by Meta (formerly Facebook), as well as a community of individual developers and companies. It is through this interface that the users will interact with the solution, from signing up to receiving and rating recommendations.

The Core Service, the Rules Service, and the Recommender Service shall be created in Java, using the Spring Boot framework. Java is a very mature programming language that is over twenty years old and has proven to be the preferred choice for many enterprise software solutions throughout the years. Spring Boot, on the other hand, is more recent but was developed to facilitate the creation of stand-alone, production-grade Spring based applications. Meaning that Spring Boot is an extension of Spring, a Java framework that provides comprehensive infrastructure support for developing Java applications.

The purpose of the Core Service is to serve as a bridge between the Graphical User Interface (GUI) and the rest of the system. The Core Service shall provide an Application Programming Interface (API) so that the users can perform their desired actions within the scope of the application through the GUI, and a WebSocket connection to exchange data in real-time, allowing notification dispatching without intervention from the GUI. In turn, the Core Service shall also communicate with the Rules Service, the Recommender Service, and the Stress Detection Service through their respective APIs. Furthermore, Core Service shall create the data structure presented in subchapter 4.4 in the open-source relational database engine PostgreSQL, which easily integrates with the Spring Boot framework.

Considering the questionnaire is an important part of the solution, making it possible to begin creating user profiles as they join the application, it shall be as detailed and complete as possible while maintaining the flexibility for adaptation and changes, not only as the solution is developed but also afterwards. Thus, instead of programmatically defining all the rules related to the questions that shall be displayed to the user and their corresponding logic, a rules engine such as Drools shall be used. Through an API, the Core Service shall be capable of requesting the necessary information to the Rules Service for the questionnaire to be displayed to the user.

The Recommender Service, as the name implies, shall be responsible for calculating the most adequate recommendations for a given user and providing that information to the Core Service so that it can be redirected to the GUI. Libraries such as MLib, from Apache Spark, or the Apache Mahout framework, are both reliable software tools for the development of a recommender system and can be integrated in the solution that shall be built in Java (Panigrahi, Lenka, & Stitipragyan 2016; Walunj & Sadafale 2013).

The Stress Detection Service is an external module, out of scope of this project, that shall receive through its API several physiological signals extracted from images of the users of the application such as pupil diameter, eye blinking, heart rate variability, and facial expressions, collected by a video plethysmography software. Then, using a trained machine learning model, shall reply to the Core Service if the received data shows signs of stress-related behavior.

Lastly, the purpose of the Message Broker is to handle an asynchronous communication between the Core Service and the Recommender Service. Generating recommendations is not linear in terms of computational complexity and relying on an HTTP communication may prove to be a shortcoming for it is not intended for the Core Service to block its execution during an HTTP request to the Recommender Service. Apache Kafka and Apache ActiveMQ are both suitable candidates for the inclusion of this component within the system.

5 Implementation

This chapter presents and describes the process involved in the implementation of the components and functionalities proposed in the Design chapter, and it is divided in subchapters representing each component. The components that required implementation as part of the solution are the Graphical User Interface, the Core Service, the Rules Service, and the Recommender Service. Albeit not implemented in the same sense as the rest of the components, the Message Broker required integration with some components of the solution and, for that reason, is also described in this chapter.

5.1 Graphical User Interface

Some of the most important characteristics in a web-based application are the ease of use, but also how visually consistent and pleasant it is. For that reason, the design of the user interface resulted from the collaboration with the stakeholders, who shared their preferences and intentions for the application regarding its visual experience.

To further facilitate this collaboration, it was used an interface design tool called Figma[8]. This tool is a web-based application that allows for multiple people to work together at the same time around the graphical design of a solution or product. Furthermore, it provides the Cascading Style Sheets (CSS) straight from the user interface, meaning that, most of the times, it was possible to obtain the right CSS for the interface components when developing the graphical user interface.

Figure 17 shows a snapshot of the Figma workspace that was used throughout the development of the graphical user interface, and Figure 18 demonstrates how the CSS can be obtained from the application.

[8] https://www.figma.com/

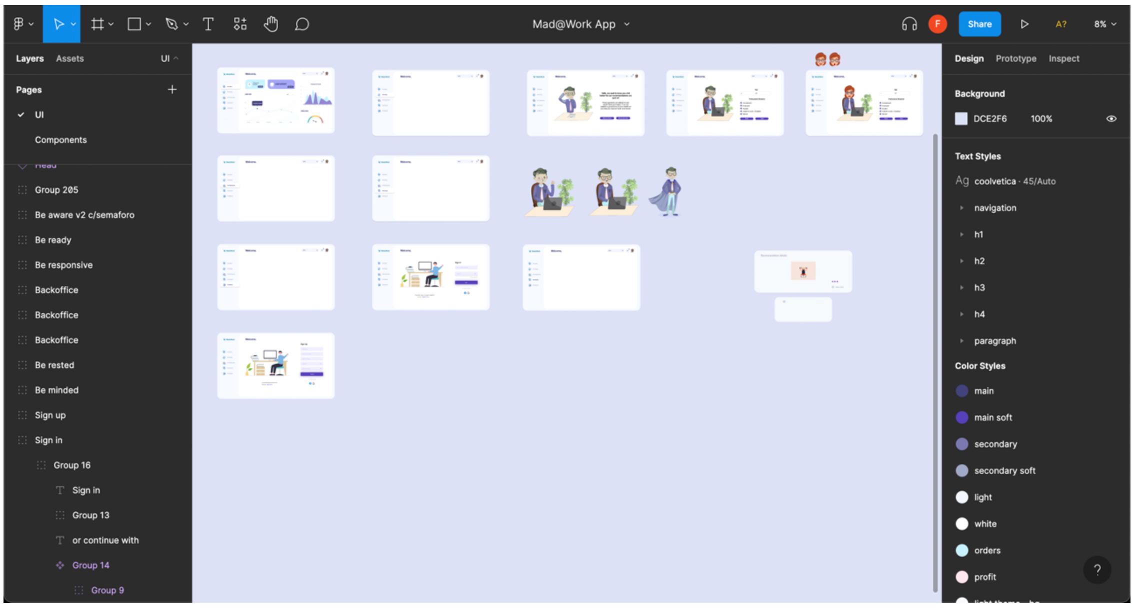
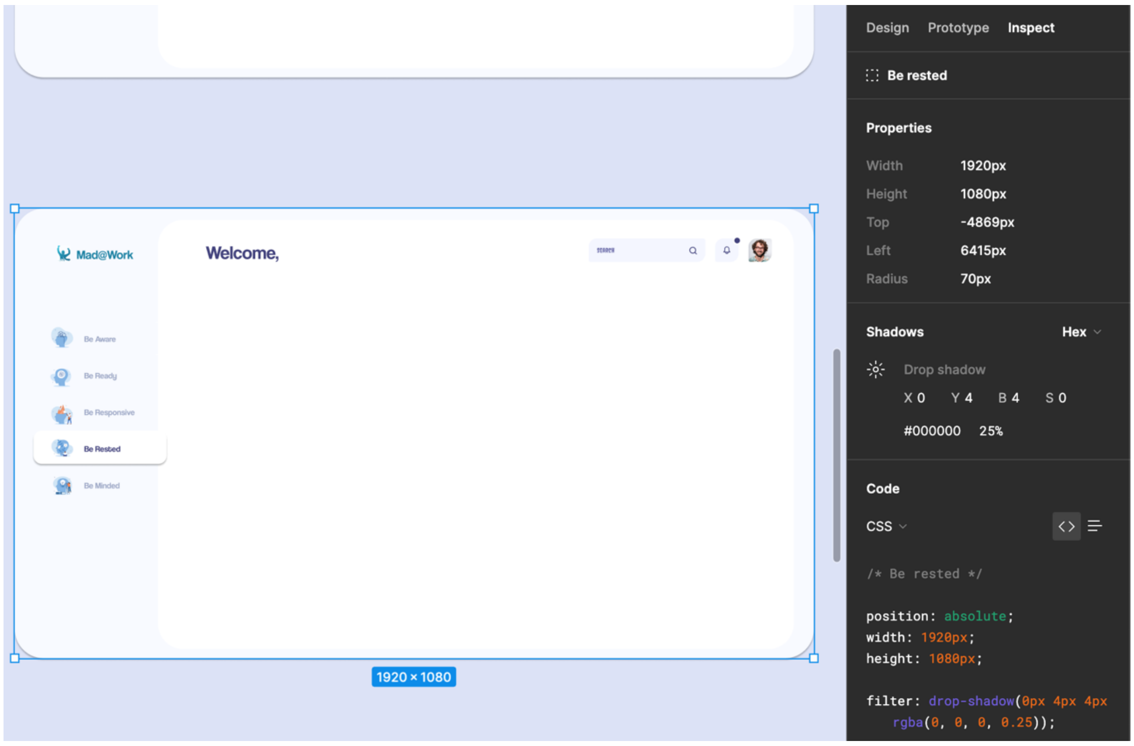


Figure 17 - Figma workspace



Be rested

Properties

Width	1920px
Height	1080px
Top	-4869px
Left	6415px
Radius	70px

Shadows Hex

Drop shadow
X 0 Y 4 B 4 S 0
#000000 25%

Code

CSS

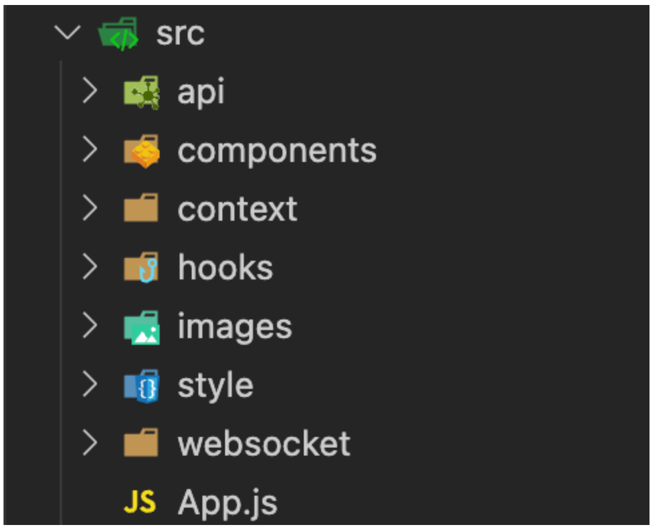
```
/* Be rested */

position: absolute;
width: 1920px;
height: 1080px;

filter: drop-shadow(0px 4px 4px
    rgba(0, 0, 0, 0.25));
```

Figure 18 - CSS properties from Figma component

Although it is possible to identify different user interface components in Figure 17, it does not represent the actual scope of the graphical user interface, meaning that not every user interface component present in the Figma workspace was implemented in the graphical user interface of the solution.

As mentioned in subchapter 4.5, the graphical user interface was developed in React, a JavaScript library. Through this library it was possible to implement the desired user interface along with the expected functionalities, which will be described ahead.

The structure for this architecture component can be divided in eight parts, as shown in Figure 19.



```
src
  > api
  > components
  > context
  > hooks
  > images
  > style
  > websocket
    App.js
```

Figure 19 - Graphical user interface structure

The file **App.js** controls the core aspects of the application such as routing, role management, and data contextualization, enabling the rest of the application components to work seamlessly with each other in a way that makes sense in terms of business domain.

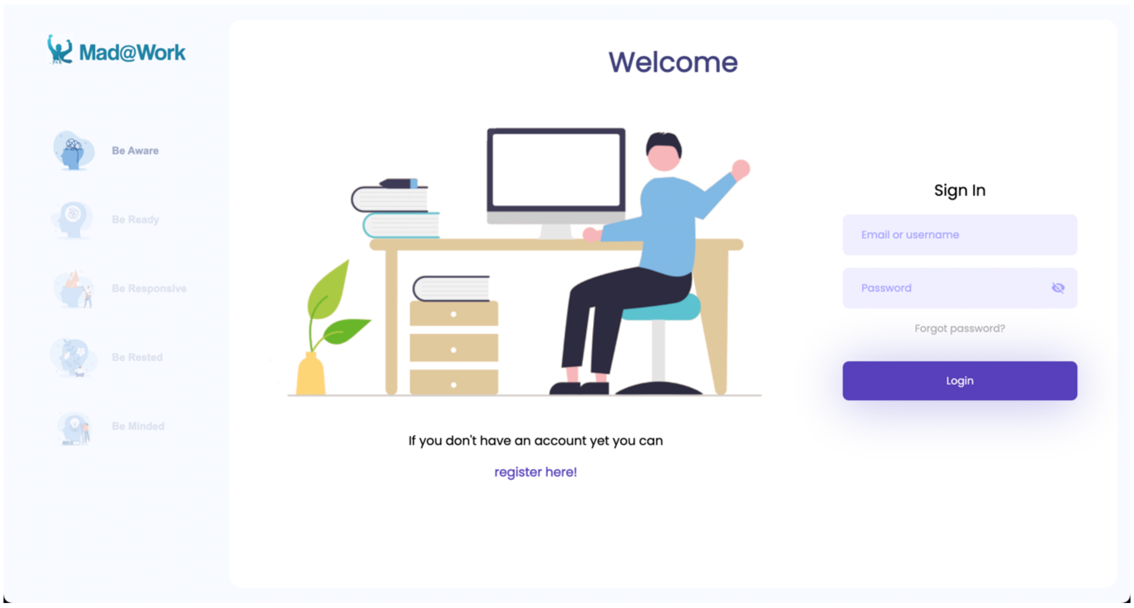
For communication, more specifically with the back-end, it is possible to see in Figure 19 two folders called **websocket** and **api**. The first folder contains a React component that uses a library called **SockJS** to establish the WebSocket connection with the back-end, and the second folder contains another React component called **Axios** that acts as an HTTP client, enabling a RESTful communication also with the back-end.

The **components** folder contains all the JavaScript files referring to the content of what is displayed in the graphical user interface. These files are composed by a mixture of JavaScript functions, HTML, and CSS. Considering the amount of styling necessary for the HTML to be presented as expected, and of images that make part of the graphical user interface, both **style** and **images** folders were created to aggregate their corresponding file types.

To accommodate the need for components to share information with each other it was created the **context** folder, containing the JavaScript files responsible for holding the data regarding user authentication and domain-wise information.

Furthermore, React provides a mechanism to use React features without having to create more components, through the usage of Hooks, allowing for components to access global states and functions. These Hooks were aggregated in the **hooks** folder.

Figure 20 shows the initial page upon landing on the application.

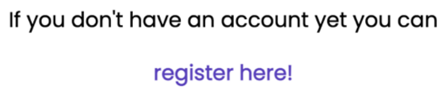


If you don't have an account yet you can

register here!

Figure 20 - Sign-in page

Through this page, users may either sign in or navigate to the sign-up page. To sign in, users must provide their username and password. For the users that do not yet own an account to sign in, it is possible for them to navigate to the sign-up page by clicking on **register here!** as highlighted in Figure 21.



If you don't have an account yet you can

register here!

Figure 21 - Link to the sign-up page from sign-in page

Furthermore, it is possible to see a list of icons on the left side of this page. Those five icons represent core graphical user interface modules, and through the usage of an opacity layer on four of them, it lets users know that in the current state of the application, the only working module is the **Be Aware** module.

In the sign-up page, users will be prompted to insert some information such as email, username, contact number, and password, having to repeat the password once to make sure it is properly inserted, as shown in Figure 22.

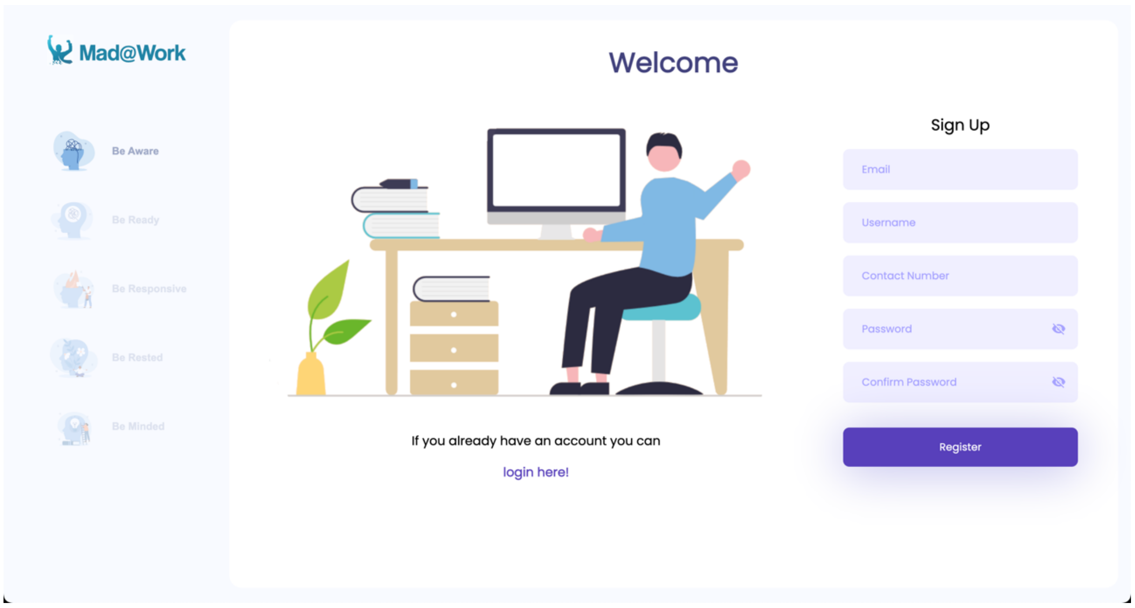


Figure 22 - Sign-up page

However, users that find themselves on the sign-up page can also navigate back to the sign-in page using the same logic previously described. In Figure 23, it is shown the link that allows users navigate to the sign-in page from the sign-up page, by clicking on **login here!**.

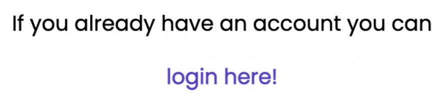
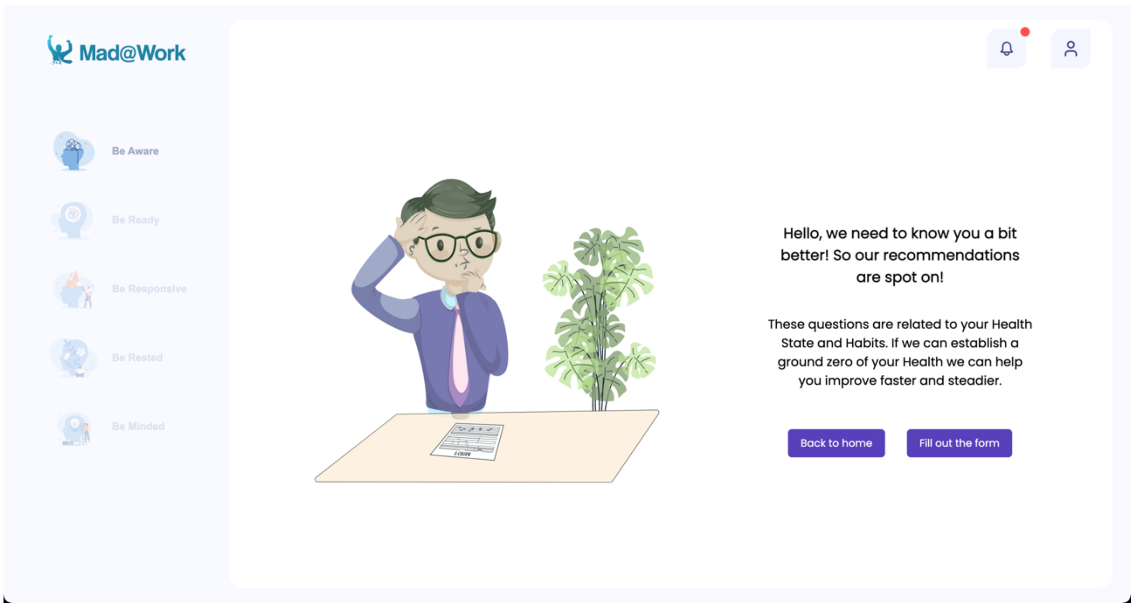


Figure 23 - Link to the sign-in page from sign-up page

After successfully signing in or signing up, users are redirected to a page with some information pointing out the importance of gathering some data related to them for the application to provide appropriate recommendations. As shown in Figure 24, in this page users can either choose to navigate to the home page or the form page.

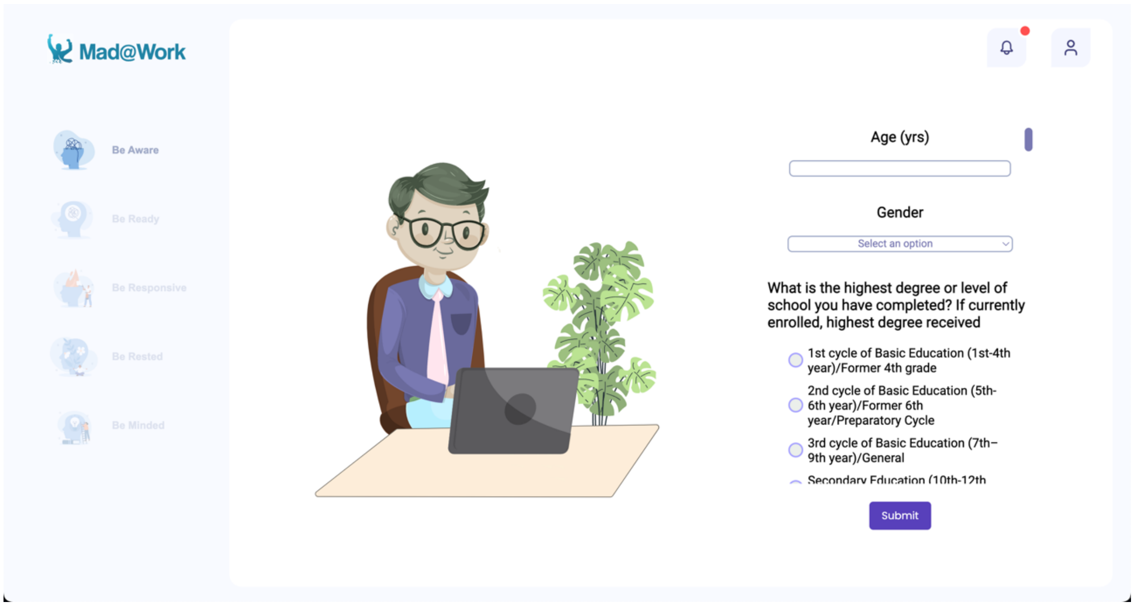


Hello, we need to know you a bit better! So our recommendations are spot on!

These questions are related to your Health State and Habits. If we can establish a ground zero of your Health we can help you improve faster and steadier.

Figure 24 - Preform page

By choosing to navigate to the form page, users are asked to answer to a vast range of questions regarding their well-being such as eating and sleeping habits, physical activity, social interaction, and past health related conditions. As shown in Figure 25, users can select one of the many options available for each question.



Age (yrs)

Gender

Select an option

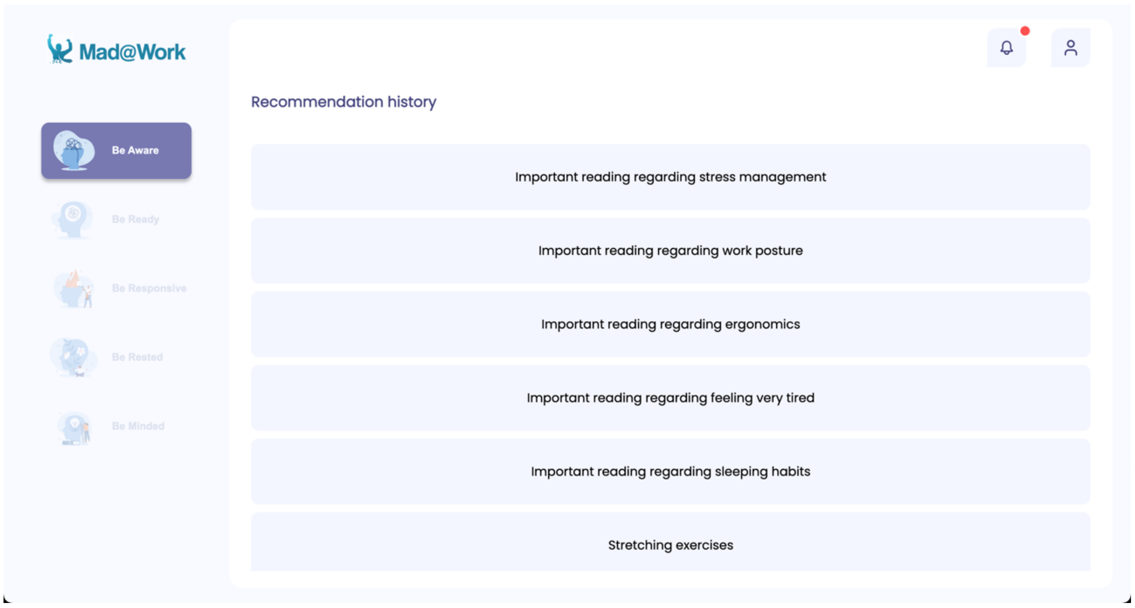
What is the highest degree or level of school you have completed? If currently enrolled, highest degree received

- 1st cycle of Basic Education (1st-4th year)/Former 4th grade
- 2nd cycle of Basic Education (5th-6th year)/Former 6th year/Preparatory Cycle
- 3rd cycle of Basic Education (7th–9th year)/General
- Secondary Education (10th-12th

Figure 25 - Form page

Every time users interact with the form by selecting one of the answers, this information is saved by the back-end. This way, if users choose to close the web application or their authentication token expires, the application can retrieve and use the information related to the questions users previously answered to. By clicking on the submit button, users acknowledge that they have provided the answers to the questions they chose to answer to and are redirected to the home page.

In the context of a web-based application focused primarily on providing tailored recommendations to its users, the home page for the application was defined as a list of past recommendations that users can interact with and learn more about them. In chapter 7, there are some observations regarding what could be done to create an even more engaging and rich home page for the application. However, in the current state of the application, Figure 26 depicts what the home page looks like.



Recommendation history

Important reading regarding stress management

Important reading regarding work posture

Important reading regarding ergonomics

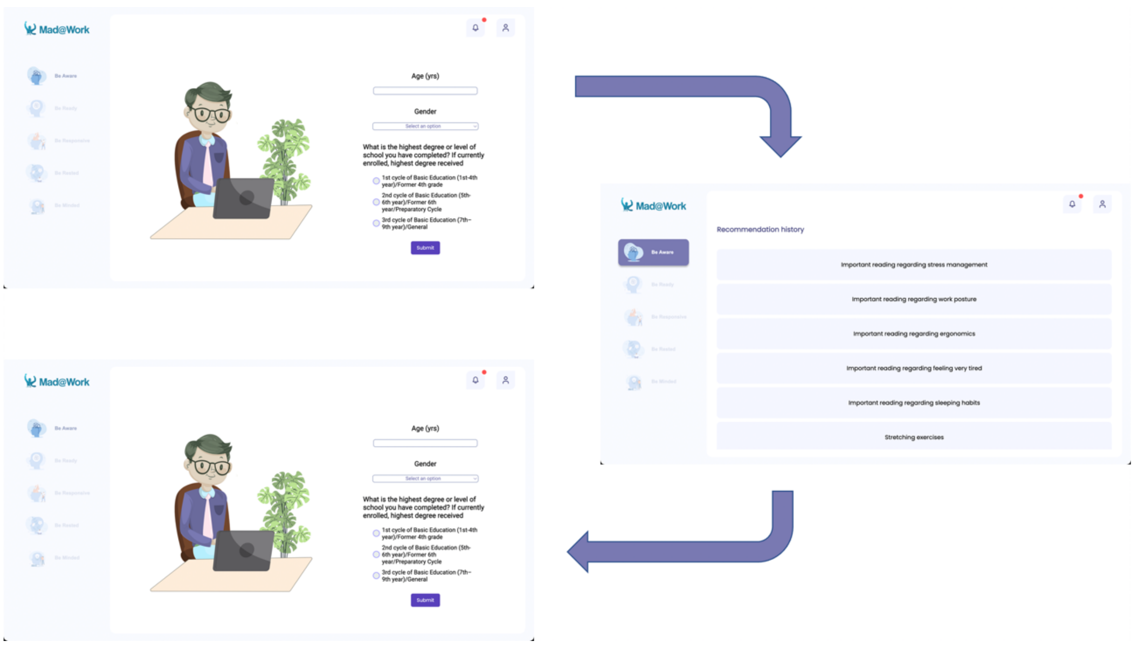
Important reading regarding feeling very tired

Important reading regarding sleeping habits

Stretching exercises

Figure 26 - Recommendation history page

There are three ways for users to navigate to the home page, although only two have been described thus far. One is by skipping the form right after singing in, and the other is by going through the form and submitting it. The third way for users to navigate to the home page is slightly more interactive, and has a role in the overall application, providing easy access to the home page while allowing for users to go back to the page they were at before, as shown in Figure 27.



Age (yrs)

Gender

Select an option

What is the highest degree or level of school you have completed? If currently enrolled, highest degree received

- 1st cycle of Basic Education (1st-4th year)/Former 4th grade
- 2nd cycle of Basic Education (5th-6th year)/Former 6th year/Preparatory Cycle
- 3rd cycle of Basic Education (7th–9th year)/General

Submit

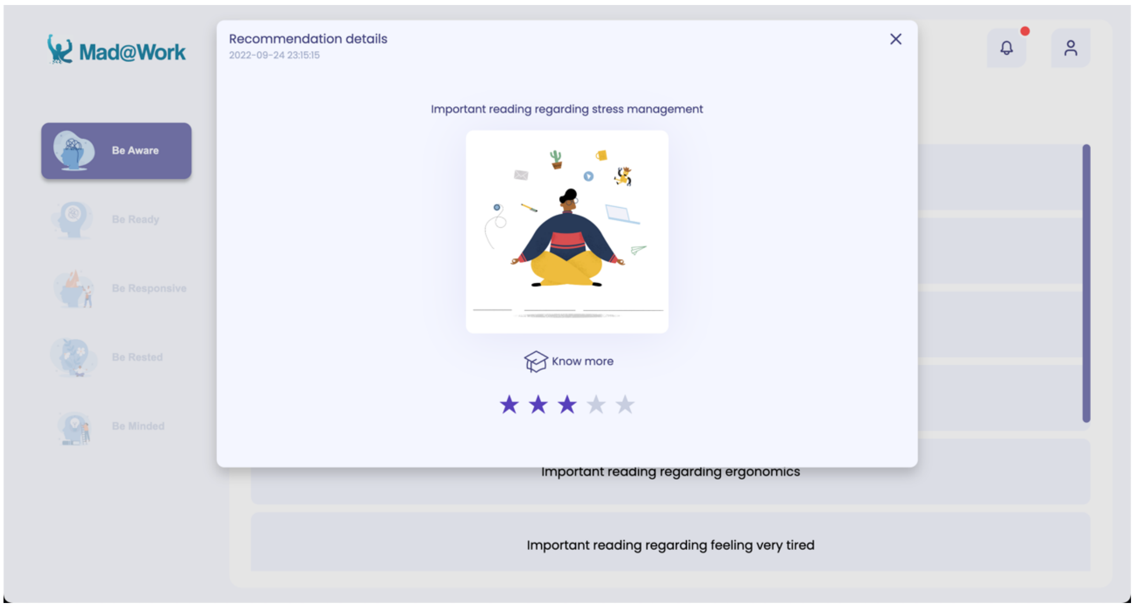
Recommendation history

- Important reading regarding stress management
- Important reading regarding work posture
- Important reading regarding ergonomics
- Important reading regarding feeling very tired
- Important reading regarding sleeping habits
- Stretching exercises

Figure 27 - Home page transition

By clicking on the **Be Aware** icon, users can navigate to the home page from anywhere within the application and go back to their previous page by clicking on the icon again.

Once in the home page, users can view all the recommendations provided by the application in the past. As shown in Figure 26, the only information users see is a brief description of what was recommended and the time at which it was recommended. However, by clicking on a recommendation from the list, a card appears and displays further details related to the recommendation that was clicked on, as represented in Figure 28.



Recommendation details
2022-09-24 23:15:15

Important reading regarding stress management

Important reading regarding ergonomics

Important reading regarding feeling very tired

Figure 28 - Recommendation card

By hovering with their mouse cursor on the **Know more** icon, users can see a list of PDF documents in which they can click on. Clicking on a document from said list opens a new tab on the browser that displays the correspondent content, as shown by Figure 29 and 30.

These documents provide advice and refer to the benefits associated with adopting the proposed behaviours.

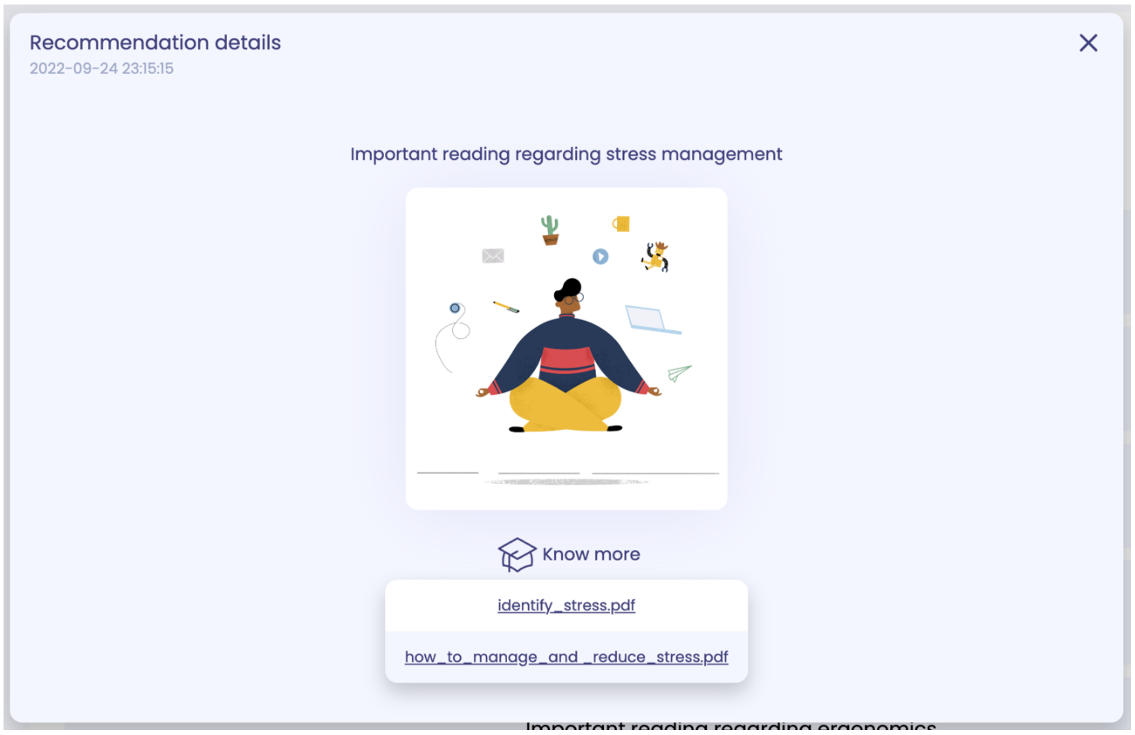
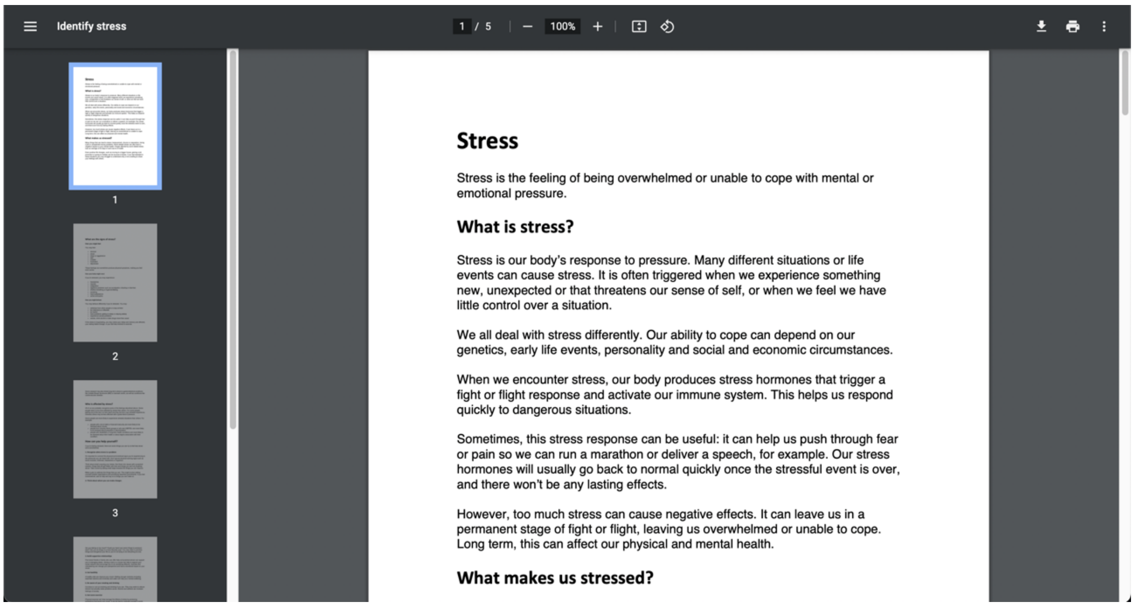

Recommendation details
2022-09-24 23:15:15
Important reading regarding stress management
Know more
identify_stress.pdf
how_to_manage_and_reduce_stress.pdf
Important reading regarding ergonomics

Figure 29 - Know more list



Stress

Stress is the feeling of being overwhelmed or unable to cope with mental or emotional pressure.

What is stress?

Stress is our body's response to pressure. Many different situations or life events can cause stress. It is often triggered when we experience something new, unexpected or that threatens our sense of self, or when we feel we have little control over a situation.

We all deal with stress differently. Our ability to cope can depend on our genetics, early life events, personality and social and economic circumstances.

When we encounter stress, our body produces stress hormones that trigger a fight or flight response and activate our immune system. This helps us respond quickly to dangerous situations.

Sometimes, this stress response can be useful: it can help us push through fear or pain so we can run a marathon or deliver a speech, for example. Our stress hormones will usually go back to normal quickly once the stressful event is over, and there won't be any lasting effects.

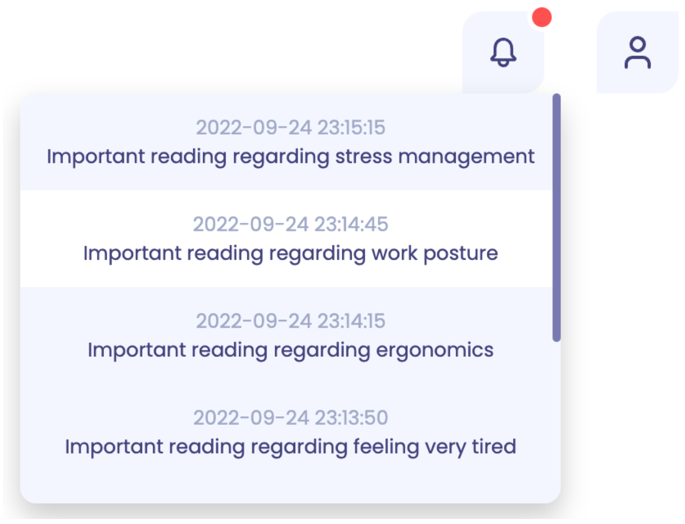
However, too much stress can cause negative effects. It can leave us in a permanent stage of fight or flight, leaving us overwhelmed or unable to cope. Long term, this can affect our physical and mental health.

What makes us stressed?

Figure 30 - Know more document

This card not only allows for a more detailed vision of the recommendation, but it also provides a rating system for users to interact with, by clicking on any of the stars at the bottom. A recommendation can be rated from one to five stars, and these ratings are taken into consideration for future recommendations.

It is possible to see, in most of the Figures related to the graphical user interface, an icon that resembles a bell and that is often associated with notifications. It is through notifications that users receive recommendations from the application. When users receive notifications, a red dot appears on the left upper corner of the icon, and by hovering their mouse on the icon they can see the latest notifications sent from the back-end, as shown in Figure 31. Once users have seen all the notifications, the red dot disappears.


2022-09-24 23:15:15
Important reading regarding stress management

2022-09-24 23:14:45
Important reading regarding work posture

2022-09-24 23:14:15
Important reading regarding ergonomics

2022-09-24 23:13:50
Important reading regarding feeling very tired

Figure 31 - Notifications

Lastly, users can opt to sign out from the application by hovering their mouse on the icon that resembles a person and clicking on the logout option as shown in Figure 32, which redirects them to the sign-in page and deletes the authentication cookies from their web browser.

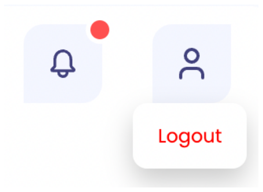


Figure 32 - Sign-out button

5.2 Message Broker

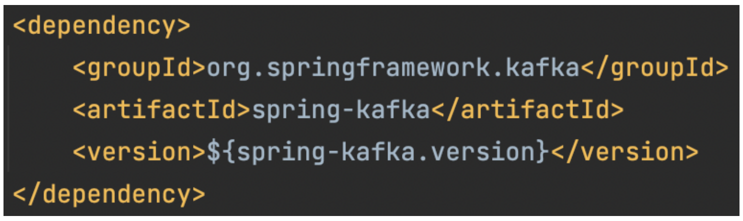
In a microservices-oriented architecture, integrations with other services sometimes lead to unexpected bottlenecks due to the blocking nature of synchronous requests. For this reason, instead of relying on just the typical HTTP connection between the services composing the solution through an API, a message broker was included, allowing for an asynchronous communication when and where necessary.

Both Kafka and ActiveMQ were eligible candidates to assume the place of a message broker in the solution. However, Kafka provides greater availability and reliability (Fu, Zhang, & Yu 2020), which led to its adoption.

Kafka is a distributed system that consists of a cluster of brokers in which is possible to configure topics. These topics can be described as a log of events and are composed by one or more partitions. Each broker is responsible for managing partitions of the existing topics and many related configurations such as replication factor and retention time (Vohra 2016).

To communicate through Kafka, it is necessary to define both producers and consumers. Producers have the responsibility of publishing messages to the Kafka topics, while consumers are those that subscribe to those topics, creating a stream of data. It is through this mechanism that an asynchronous communication is provided.

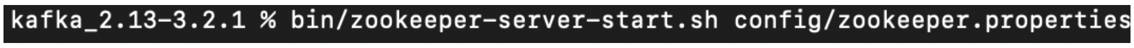
As mentioned in subchapter 4.5, requests from Core Service to Recommender Service should not block the execution workflow of the solution. Considering that, Kafka was integrated in the solution by adding the necessary dependency to the corresponding services, as shown in Figure 33.



```xml
<dependency>
    <groupId>org.springframework.kafka</groupId>
    <artifactId>spring-kafka</artifactId>
    <version>${spring-kafka.version}</version>
</dependency>
```

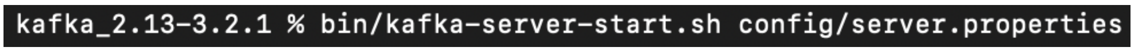
Figure 33 - Kafka maven dependency

Then, a Kafka broker was launched by running executables on premises through the command line, and both core-service and recommender-service topics were created, as shown by Figures 34 and 35, and Code snippets 1 and 2, respectively.



```
kafka_2.13-3.2.1 % bin/zookeeper-server-start.sh config/zookeeper.properties
```

Figure 34 - Start Zookeeper server



```
kafka_2.13-3.2.1 % bin/kafka-server-start.sh config/server.properties
```

Figure 35 - Start Kafka server

```java
@Configuration
public class KafkaConfig {

    @Bean
    public NewTopic createCoreServiceTopic() {
        return new NewTopic("core-service-topic", 1, (short) 1);
    }
}
```

Code snippet 1 - Core Service topic creation

```java
@Configuration
public class KafkaConfig {

    @Bean
    public NewTopic createRecommenderServiceTopic() {
        return new NewTopic("recommender-service-topic", 1, (short) 1);
    }
}
```

Code snippet 2 - Recommender Service topic creation

Lastly, consumers and producers were defined in both services, allowing the asynchronous exchange of data, as shown by Code snippets 3 to 6.

```java
@KafkaListener(topics = "core-service-topic", groupId = "core-service-consumer-
group")
public void listen(@Header(KafkaHeaders.RECEIVED_TOPIC) String topicName, @Payload
String message) {

    log.info("Received message in topic {}: {}", topicName, message);

    …
}
```

Code snippet 3 - Core Service Kafka consumer

```java
@Slf4j
@Service
@RequiredArgsConstructor
public class ProducerService {

    private final KafkaTemplate<String, String> kafkaTemplate;

    public void sendMessage(String message) {
        kafkaTemplate.send("recommender-service-topic", message);
        log.info("Sent message {} to topic recommender-service-topic", message);
    }
}
```

Code snippet 4 - Core Service Kafka producer

```
@KafkaListener(topics = "recommender-service-topic", groupId = "recommender-service-
consumer-group")
public void listen(@Header(KafkaHeaders.RECEIVED_TOPIC) String topicName, @Payload
String message) {

    log.info("Received message in topic {}: {}", topicName, message);

    …
}
```

Code snippet 5 - Recommender Service Kafka consumer

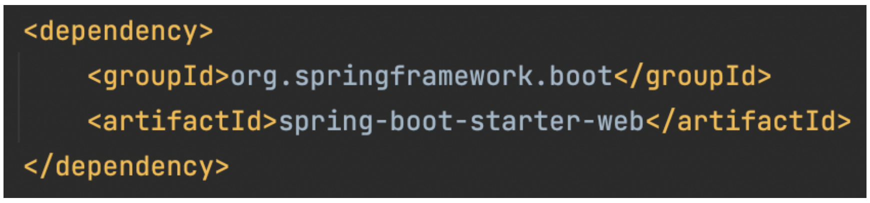
```
@Slf4j
@Service
@RequiredArgsConstructor
public class ProducerService {

    private final KafkaTemplate<String, String> kafkaTemplate;

    private final ObjectMapper objectMapper = new ObjectMapper();

    public void sendMessage(RecommenderServiceResponseDto
recommenderServiceResponseDto) {
        try {
            String message =
objectMapper.writeValueAsString(recommenderServiceResponseDto);
            kafkaTemplate.send("core-service-topic", message);
            log.info("Sent message {} to topic core-service-topic", message);
        } catch (Exception e) {
            log.error("Failed to send message to core-service-topic: {}",
e.getMessage());
        }
    }
}
```

Code snippet 6 - Recommender Service Kafka producer

5.3 Core Service

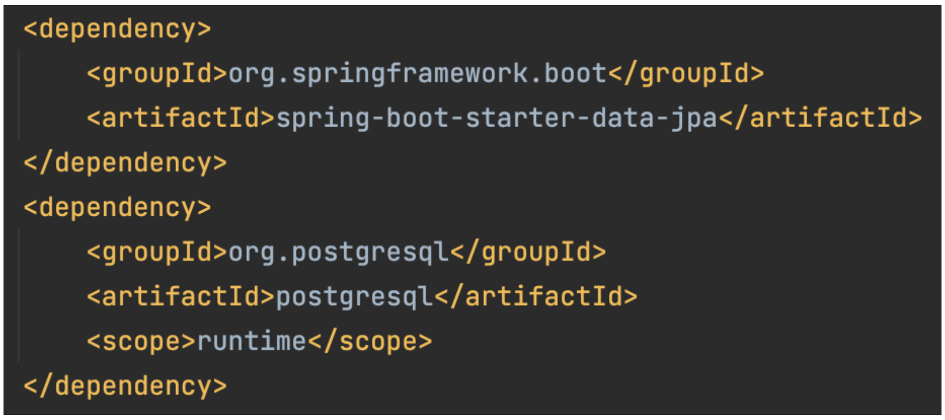
In the architecture proposed in subchapter 4.5, Core Service is presented as the point of contact between the frontend and the backend components, meaning that every request made from the Graphical User Interface (GUI) is received and answered directly by the Core Service. This way, it was possible to serve the interaction between users and application while segregating the logic related to recommendations and rules.

This service is responsible not only for supporting all the functionalities described in subchapter 5.1, but also for creating the data structure described in subchapter 4.4, handling authentication and authorization, managing WebSocket sessions, and establishing an asynchronous connection through Kafka messaging. For this, it was necessary to add the dependencies shown in Figures 36 to 40.



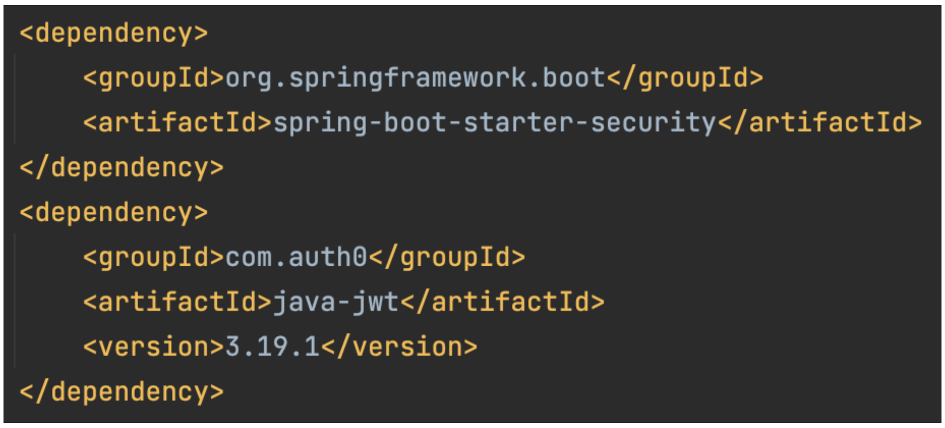
```xml
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

Figure 36 - Web maven dependency



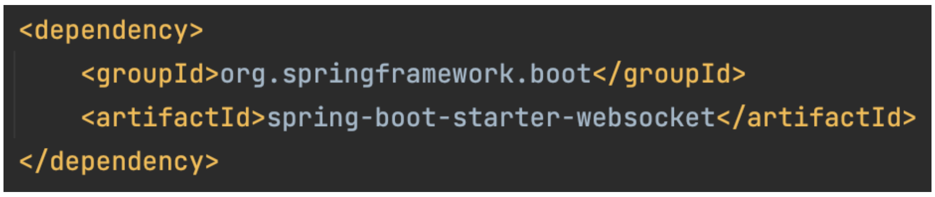
```xml
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
    <groupId>org.postgresql</groupId>
    <artifactId>postgresql</artifactId>
    <scope>runtime</scope>
</dependency>
```

Figure 37 - Database maven dependencies



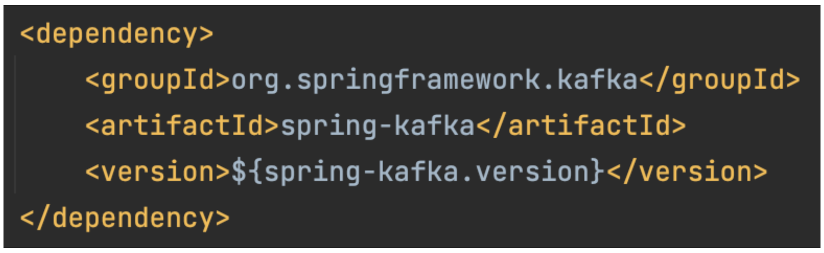
```xml
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
    <groupId>com.auth0</groupId>
    <artifactId>java-jwt</artifactId>
    <version>3.19.1</version>
</dependency>
```

Figure 38 - Security maven dependencies



```xml
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-websocket</artifactId>
</dependency>
```

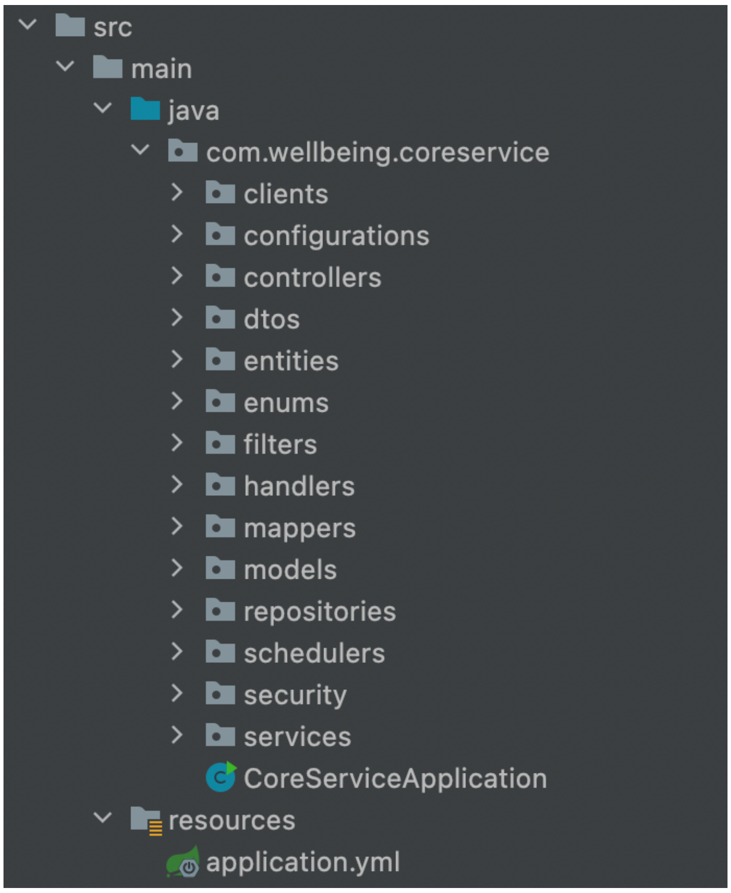
Figure 39 - WebSocket maven dependency



```
<dependency>
    <groupId>org.springframework.kafka</groupId>
    <artifactId>spring-kafka</artifactId>
    <version>${spring-kafka.version}</version>
</dependency>
```

Figure 40 - Kafka maven dependency

As for the structure, Core Service follows a controller-service-repository approach, albeit composed by other components, as shown by Figure 41.



```
src
  main
    java
      com.wellbeing.coreservice
        clients
        configurations
        controllers
        dtos
        entities
        enums
        filters
        handlers
        mappers
        models
        repositories
        schedulers
        security
        services
        CoreServiceApplication
    resources
      application.yml
```

Figure 41 - Core service structure

The file **CoreServiceApplication.java** contains the *Main* method, along with other Spring Boot annotations responsible for starting the service with all the necessary resources, and the file **application.yml** contains information regarding the application that is used by both automatic and manual configurations, such as which port to run the service on, database connection address, username, and password, as well as the addresses for the Rules Service, the Recommender Service, and Kafka bootstrap servers.

The **configurations** folder holds not only the REST client configuration, but also both Kafka and WebSocket configuration. The **clients** folder contains the interfaces that are used to define the REST client endpoints that Core Service communicates with.

Regarding the data structure, all entities described in subchapter 4.4 are represented as Spring Boot entities in the **entities** folder. By doing so, and by defining their corresponding repositories in the **repositories** folder, Spring Boot automatically created the data structure in a PostgreSQL database engine instance.

Considering that sometimes it is not desirable to return all data related to an entity, it is necessary to map it to another object containing only the information that is relevant. Therefore, both the **dtos** and **mappers** folders were created to aggregate their corresponding Java files. In the same way, not all Java objects that represent a data structure, or part of it, identify as an entity. Those that do not, belong to either the **models** or **enums** folder.

To support the functionalities provided by the GUI, Core Service required an API. This API is defined within the **controllers** folder, and all the business logic that is not related to generating recommendations or processing rules is contained within the **services** folder, along with both Kafka consumer and producer.

The Core Service API is composed by the following endpoints:

- POST /login

  This is the endpoint responsible for allowing a user to sign in by providing a *username* and *password*. If the user successfully signs in, both an *access token* and a *refresh token* are returned by the endpoint in the form of a *JSON Web Token*. Otherwise, the endpoint returns an HTTP 403 code.

- POST /logout

  After signing in, users have the possibility to sign out through this endpoint. By doing so, any authentication token in their possession is invalidated and can no longer be used.

- POST /user

  Users must sign up prior to signing in. This endpoint allows for any user to register in the application by providing an *e-mail*, a *username*, a *password*, and a *confirmation password*, along with the optional field *contact*.

- GET /refresh-token

  When users sign in, they are given an *access token* and a *refresh token*.

  The *access token* contains user related information that are relevant in terms of security, for example, such as the user role. This way, the GUI can implement a role-based access to the interface resources. However, for security reasons, the *access token* should have a short validity, and to avoid having users signing in every other minute, they can use the *refresh token* to request a new *access token*, as long as the *refresh token* is still valid.

The *refresh token* is saved in the users' browser as a secure cookie and has a greater validity, unlike the *access token*. To use it, when users make a request through the GUI to the Core Service, there is an interceptor that checks if an *access token* exists and whether it is still valid. If condition is not true, the interceptor makes a request to this endpoint to get a new *access token* and proceeds with the initial request done by the user.

- GET /recommendation-history

  This endpoint receives the *access token* from the user making the request and returns all past recommendations provided by the Recommender Service to the corresponding user.

- POST /recommendation/{recommendationId}/rating

  Users can rate the recommendations provided to them, contributing to the creation and development of a user profile that helps the Recommender Service generating recommendations that are best suited for them. This endpoint receives the *access token* from the user making the request, as well as the identifier of the recommendation being rated, and the corresponding rating given by the user.

- GET /questions

  This endpoint receives the *access token* from the user that is signed in and currently in the Form page, in the GUI, and returns the questions composing this page depending on defined business rules. For this, Core Service communicates with the Rules Service to deliver the right questions to users.

- POST /answer

  For every option that users choose in the Form page, this endpoint is called by the GUI and receives both the *access token* of the users selecting the answers, as well as the chosen answer and the question that the answer refers to, allowing for the Core Service to save this information.

- GET /notifications

  This is the endpoint responsible for getting the notifications of signed in users. Right after users signing in, this endpoint is called with the *access token* of the signed user and returns the collection of past notifications.

However, not all communication with the Core Service is done through the aforementioned endpoints.

Although there is an endpoint to get the notifications right after users sign in on the application, it was necessary to adopt a mechanism to deliver notifications in real-time without having to constantly make HTTP requests from the GUI to the Core Service. This was achieved by using WebSockets, and it is defined in the **handlers** folder. Once users sign in, the GUI is responsible for establishing a WebSocket session with the Core Service, and Core Service is responsible for creating a data structure holding the identifier of the established WebSocket session, along with the user identifier it belongs to and a timestamp.

This service also contains a scheduler, that can be found in the **schedulers** folder, responsible for getting all the active WebSocket sessions and, depending on if the time at which the scheduler is running is greater than the timestamp associated with the sessions, for sending a message through Kafka to the Recommender Service requesting for a recommendation for each user identifier associated with the sessions. As soon as the Recommender Service replies, also through Kafka, the message is caught by the consumer of Core Service that sends a notification to the corresponding users through their WebSocket sessions, containing the generated recommendations.

Regarding users' authentication and authorization, both **security** and **filters** folders aggregate web security configurations, as well as authentication and authorization filters, respectively.

Web security configurations include which authentication manager and password encoder to be used, but also which endpoints are allowed to be called without authentication, such as:

- POST /login
- POST /logout
- POST /user
- GET /refresh-token

The authentication filter is executed every time users attempt to sign in. First, the *username* and *password* provided by users is taken and checked against the information that exists in the user database. Then, if the validations pass, the authentication filter returns a secure HTTP cookie with a *JSON Web Token* called *refresh token* and another *JSON Web Token* called *access token* in the body of the response to the sign in request.

Lastly, the authorization filter makes sure that the protected endpoints can only be accessed by users that own a valid *access token*. When users request for their recommendation-history through the GUI, for example, this filter intercepts the request that Core Service received and verifies the authenticity of the *access token* before letting Core Service process the request.

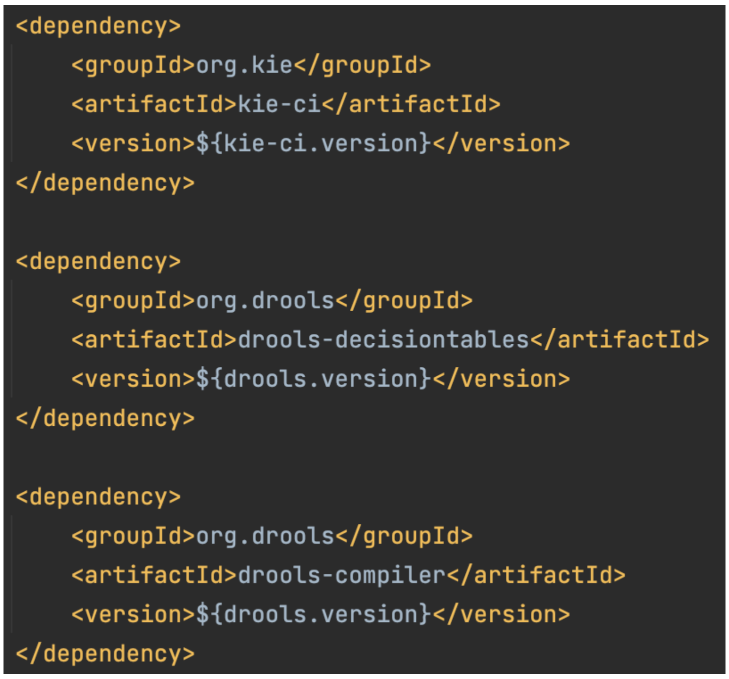
5.4 Rules Service

As mentioned in chapter 4, the solution required the inclusion of a questionnaire for users to reply to. This questionnaire could be implemented in a more traditional way, hard coding the questions and the related dependencies resulting from users answers to those questions, or in a more generic, configurable, and approachable way that would allow not only developers but also non-developers to change them. For this reason, a rules management system was used.

Rules management systems provide mechanisms to segregate business logic from application logic, making it easier to scale and maintain as business logic evolves (Velzen 2018). It is possible to represent the business logic as something that happens when a set of constraints is fulfilled, as exemplified by expression 14:

$$\text{WHEN glass\_of\_water AND empty THEN refill\_with\_water} \tag{14}$$

Drools [9] is not the only rules management system that could be used to power this configurability and approachability. However, being open-source and easily integrable with Java made it the perfect candidate for this solution. Figure 42 shows all the dependencies required to integrate Drools with a Spring Boot project.



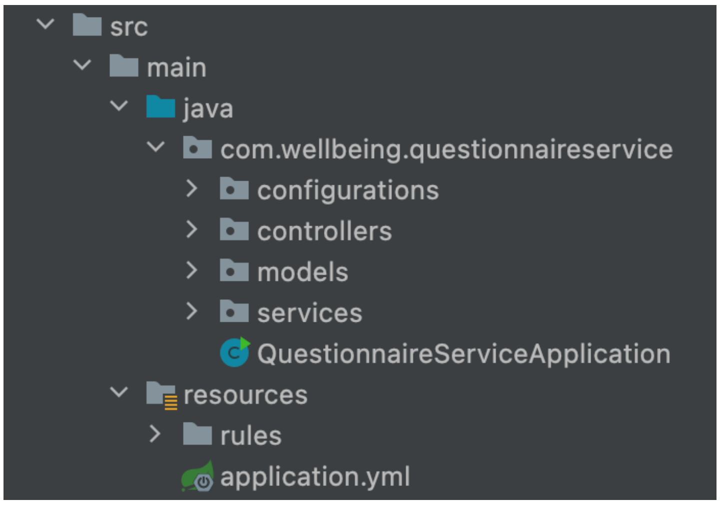
```
<dependency>
    <groupId>org.kie</groupId>
    <artifactId>kie-ci</artifactId>
    <version>${kie-ci.version}</version>
</dependency>

<dependency>
    <groupId>org.drools</groupId>
    <artifactId>drools-decisiontables</artifactId>
    <version>${drools.version}</version>
</dependency>

<dependency>
    <groupId>org.drools</groupId>
    <artifactId>drools-compiler</artifactId>
    <version>${drools.version}</version>
</dependency>
```

Figure 42 - Drools maven dependencies

Being part of a group of micro-services that communicate through HTTP, this service also required the spring-boot-starter-web dependency, which provides the core infrastructure for a web service to run.

In Figure 43, it is possible to see how the Rules Service was structured, containing two files and five folders.

[9] https://www.drools.org/



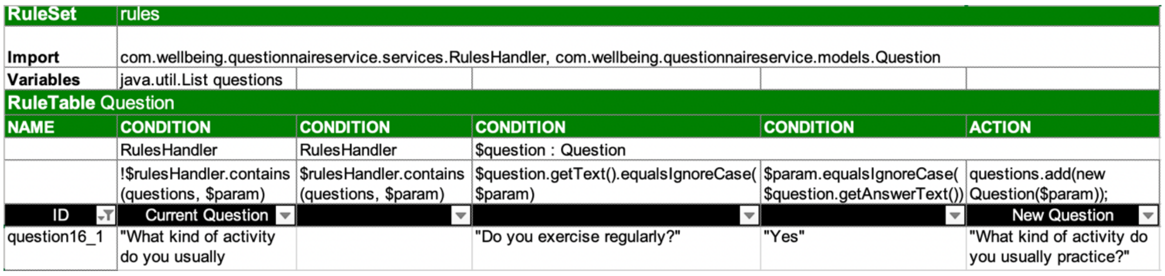
```
src
  main
    java
      com.wellbeing.questionnaireservice
        configurations
        controllers
        models
        services
        QuestionnaireServiceApplication
    resources
      rules
      application.yml
```

Figure 43 - Rules Service structure

The file **QuestionnaireServiceApplication.java** holds the method responsible for starting the service with all the necessary resources, and the file **application.yml** contains the data that is used by both automatic and manual configurations related to the service, such as exposed port and location of business rules. In turn, the **rules** folder contains the business rules specified in the **application.yml** file.

In the context of this application, business rules are directly related to questionnaire configurations.

Keeping in mind the configurability and approachability that Drools provides, its ability to read and comprehend spreadsheets allowed to write the business rules in an XLS file format, as shown in Figure 44.



RuleSet	rules				
Import	com.wellbeing.questionnaireservice.services.RulesHandler, com.wellbeing.questionnaireservice.models.Question				
Variables	java.util.List questions				
RuleTable Question					
NAME	**CONDITION**	**CONDITION**	**CONDITION**	**CONDITION**	**ACTION**
	RulesHandler	RulesHandler	$question : Question		
	!$rulesHandler.contains (questions, $param)	$rulesHandler.contains (questions, $param)	$question.getText().equalsIgnoreCase($param)	$param.equalsIgnoreCase($question.getAnswerText())	questions.add(new Question($param));
ID	Current Question				New Question
question16_1	"What kind of activity do you usually		"Do you exercise regularly?"	"Yes"	"What kind of activity do you usually practice?"

Figure 44 - Business rules sample

It is possible to see in Figure 44 that the spreadsheet follows a specific format. However, it does not deviate from the fact that it is simpler to understand and maintain when comparing to having the same logic written in code.

In the **configurations** folder, it is possible to find the Spring Boot Bean that initializes the Drools container. It is through this configuration that the Rules Service provides Drools with the business rules, allowing it to build the necessary knowledge base, as displayed by Code snippet 7.

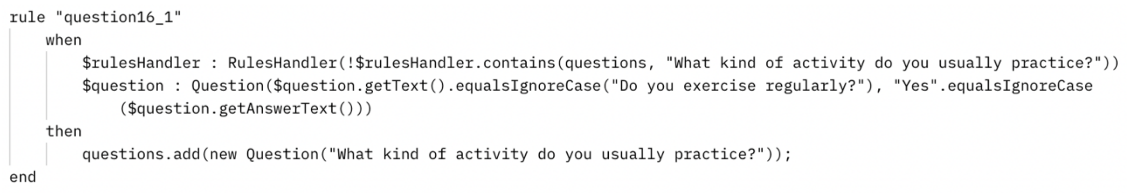
```
@Bean
public KieContainer getKieContainer() {

    KieFileSystem kieFileSystem = kieServices.newKieFileSystem();

kieFileSystem.write(ResourceFactory.newClassPathResource(questionsRulesFilePath));

kieFileSystem.write(ResourceFactory.newClassPathResource(recommendationPropertiesRule
sFilePath));

    KieBuilder kb = kieServices.newKieBuilder(kieFileSystem);
    kb.buildAll();

    SpreadsheetCompiler spreadsheetCompiler = new SpreadsheetCompiler();
    Resource resource = ResourceFactory.newClassPathResource(questionsRulesFilePath);
    String drl = spreadsheetCompiler.compile(resource, InputType.XLS).
            replace("\\\\n", System.lineSeparator()).
            replace("TRUE", "true").
            replace("FALSE", "false");
    log.info("{}", drl);

    KieModule kieModule = kb.getKieModule();
    KieContainer kieContainer =
kieServices.newKieContainer(kieModule.getReleaseId());
    KieBase kieBase = kieContainer.getKieBase();

    log.info("KieBase created: {}", kieBase.getKiePackages());

    return kieContainer;
}
```

Code snippet 7 - Drools container configuration

In addition to initializing the Drools container, the Rules Service provides an API so that other services belonging to the architecture of the solution can make requests over HTTP. This API is defined within the **controllers** folder and boils down to just two endpoints:

- GET /drlFile
- POST /trigger-question-rule

The first endpoint is responsible for taking the rules spreadsheet located in the **rules** folder and parsing it to a specific format that Drools is capable of processing, allowing for a more technical perspective on the business rules logic. Considering the business rules sample from Figure 44, it is possible to demonstrate the response from the first endpoint regarding those same rules, through Figure 45.



```
rule "question16_1"
    when
        $rulesHandler : RulesHandler(!$rulesHandler.contains(questions, "What kind of activity do you usually practice?"))
        $question : Question($question.getText().equalsIgnoreCase("Do you exercise regularly?"), "Yes".equalsIgnoreCase
            ($question.getAnswerText()))
    then
        questions.add(new Question("What kind of activity do you usually practice?"));
end
```

Figure 45 - Parsed business rules sample

Simplifying the rule **question16_1** in Figure 45, we get the expression 15:

> WHEN list of questions does not contain the question “What kind of activity do you usually practice?”
> AND current question is equal to “Do you exercise regularly?”
> AND current question is answered with “Yes”
> THEN add question “What kind of activity do you usually practice?” to list of question (15)

The second endpoint, however, relies on the files within the folders **services** and **models** to process and deliver a response to any request. This endpoint accepts a list of Java objects of type Question, initializes a Drools container session, and for each object in the list it injects said object as a variable in the parsed business rules and fires them all, updating the original list before returning it. For every rule, there is a course of action if all the conditions are met. In the context of this application, the course of action often results in the addition or removal of a question from the list of objects of type Question that users can reply to through the graphical user interface. Both the Question Java object and the processing of business rules can be represented by Code snippets 8 and 9, respectively.

```java
@Data
@AllArgsConstructor
@NoArgsConstructor
@Builder
public class Question {

    private String text;
    private String answerText;
    private String answerType;
    private List<String> answerOptions;
    private Integer order;

    public Question(String text) {
        this.text = text;
        this.answerText = null;
        this.answerType = null;
        this.answerOptions = null;
        this.order = null;
    }
}
```

Code snippet 8 - Question Java object

```java
public List<Question> triggerQuestionRule(List<Question> questions) {

    KieSession kieSession = kieContainer.newKieSession();

    kieSession.setGlobal("questions", questions);
    kieSession.setGlobal("recommendationProperties", new ArrayList<>());
    kieSession.insert(rulesHandler);

    List<Question> updatedQuestions = new ArrayList<>(questions);

    updatedQuestions.forEach(question -> {
        kieSession.insert(question);
        kieSession.fireAllRules();
    });

    if (CollectionUtils.isEmpty(questions)) kieSession.fireAllRules();

    kieSession.dispose();

    return questions;
}
```

Code snippet 9 - Processing of business rules

5.5 Recommender Service

Recommendations have a significant impact on the solution, for its main objective is to recommend healthy habits to its users.

Considering that the stakeholders involved in the development this solution are health-related professionals, the rationale behind which recommendations to include in the solution was conducted mostly by them, and posteriorly discussed in the several meetings that were held throughout the development period.

As described in chapter 5.1, and as shown by Figures 29 and 30, each recommendation has an associated collection of documents, also provided by the stakeholders, so that users can explore the recommendations they receive. Both the recommendations and the collection of documents associated with each one of them is expected to grow over time. However, at the time of deliverance of this document, the recommendations included in the solution are as follows:

- Physical exercise during 10 minutes
- Physical exercise during 5 minutes
- Stretching exercises
- Muscle activation exercises
- Important reading regarding time management
- Take a break to chat with a colleague
- Important reading regarding personal care
- Time management tips

- Important reading regarding what to drink
- Important reading regarding reducing unhealthy drinks
- Important reading regarding stress reduction
- Important reading regarding resilience
- Important reading regarding sleeping habits
- Important reading regarding work posture
- Important reading regarding ergonomics
- Important reading regarding stress management
- Important reading regarding feeling very tired

For this reason, and considering the architecture presented in subchapter 4.5, a micro-service was developed to address the business requirements related to recommendations.

For recommendations to be provided there must be at least one filtering technique in place. Subchapter 3.1 explains through an analytical process which filtering technique should be used, given the described imposed limitations, concluding that the content-based technique is the most appropriate. This type of filtering technique relies on the similarities between recommendations and users' interests. Therefore, to adopt a content-based filtering technique, it was necessary to define the following recommendation properties:

- Social interaction
- Nutrition
- Health condition
- Physical activity
- Sleep
- Mental health
- Individual

These properties represent characteristics of the recommendations being provided and allows for the calculation of similarities between recommendations and users. Table 12 represents how properties and recommendations associate with each other.

Table 12 - Recommendations and properties association

Recommendation	Properties
Physical exercise during 10 minutes	Individual, Physical activity, Sleep, Health condition
Physical exercise during 5 minutes	Individual, Physical activity, Sleep, Health condition
Stretching exercises	Individual, Physical activity, Sleep, Health condition

Muscle activation exercises	Individual, Physical activity, Sleep, Health condition
Important reading regarding time management	Social interaction, Health condition, Mental health
Take a break to chat with a colleague	Social interaction, Health condition, Mental health
Important reading regarding personal care	Social interaction, Health condition, Mental health
Time management tips	Social interaction, Health condition, Mental health
Important reading regarding what to drink	Individual, Sleep, Health condition, Nutrition
Important reading regarding reducing unhealthy drinks	Individual, Sleep, Health condition, Nutrition
Important reading regarding stress reduction	Social interaction, Sleep, Health condition, Mental health
Important reading regarding resilience	Social interaction, Physical activity, Sleep, Health condition, Mental health
Important reading regarding sleeping habits	Physical activity, Sleep, Health condition, Mental health
Important reading regarding work posture	Individual, Physical activity, Sleep, Mental health
Important reading regarding ergonomics	Individual, Physical activity, Sleep, Mental health
Important reading regarding stress management	Individual, Physical activity, Sleep, Mental health
Important reading regarding feeling very tired	Individual, Physical activity, Sleep, Mental health

Users' interests are also represented by the same properties. However, the solution does not know which properties are associated to each user by default. Instead, as users interact with the application through the Graphical User Interface, they can rate recommendations. By rating a recommendation positively, the application is responsible for assigning the properties of that recommendation to the users that rated it that way. Only then, it becomes possible to calculate similarities between both recommendations and those users.

Similarly, when users get recommendations that do not pleases them and decide to rate such recommendations negatively, some of the properties in common are removed from users' interests to avoid similar future recommendations.

Thus, within the context of a Spring Boot application, including the Apache Spark [10]open-source library allowed for the implementation of a content-based recommender service through the APIs that Apache Spark provides.

Like Core Service and Drools Service, to be able to serve an HTTP communication, Recommender Service required the spring-boot-starter-web dependency, which provides the core infrastructure for a web service to run. As for Apache Spark, to be able to use it was necessary to add the dependencies shown in Figure 46.

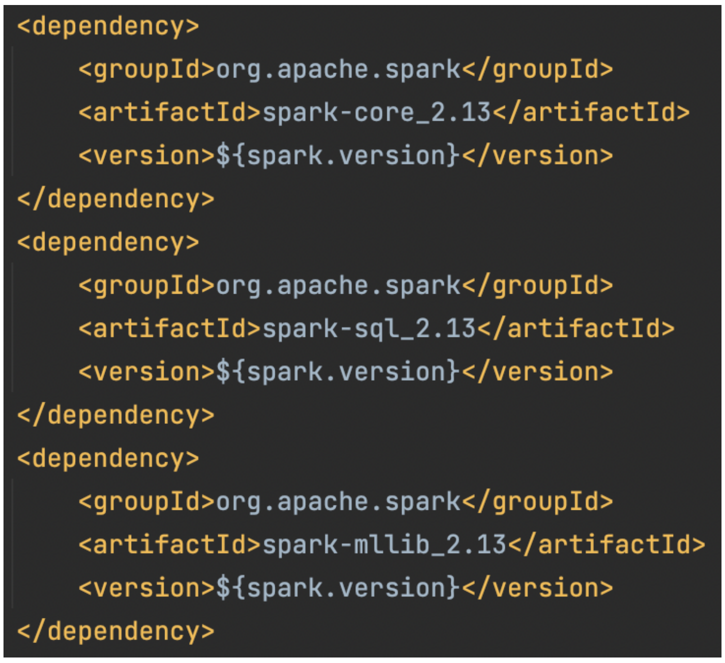
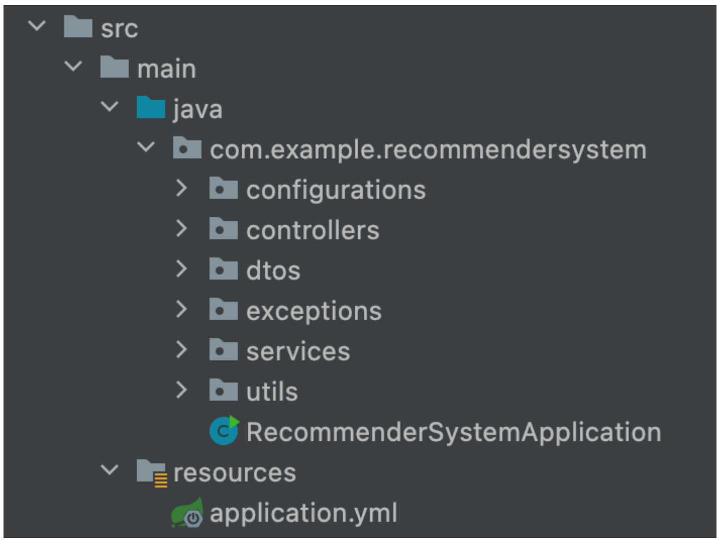

```xml
<dependency>
    <groupId>org.apache.spark</groupId>
    <artifactId>spark-core_2.13</artifactId>
    <version>${spark.version}</version>
</dependency>
<dependency>
    <groupId>org.apache.spark</groupId>
    <artifactId>spark-sql_2.13</artifactId>
    <version>${spark.version}</version>
</dependency>
<dependency>
    <groupId>org.apache.spark</groupId>
    <artifactId>spark-mllib_2.13</artifactId>
    <version>${spark.version}</version>
</dependency>
```

Figure 46 - Apache Spark maven dependencies

Recommender Service follows a similar structure to the other services composing the solution, as shown by Figure 47.

[10] https://spark.apache.org/



```
src
  main
    java
      com.example.recommendersystem
        configurations
        controllers
        dtos
        exceptions
        services
        utils
        RecommenderSystemApplication
    resources
      application.yml
```

Figure 47 - Recommender Service structure

The file **RecommenderSystemApplication.java** is responsible for starting the service with all the necessary resources, and the file **application.yml** contains automatic spring related configurations but also data used to manually configure other components within the service, such as data source connection and Spark session.

The **configurations** folder aggregates the files responsible for providing the data source connection defined in **application.yml**, creating a Kafka topic, and starting a Spark session.

Although the communication with the Recommender Service happens mostly through Kafka, the service provides an API, allowing for HTTP requests if necessary. This API is defined within the **controllers** folder and boils down to a single endpoint:

- GET /recommendation

This endpoint relies on the files within the folders **services**, **dtos**, **exceptions**, and **utils,** to process and deliver a response to any request. Similarly, both Kafka consumer and producer are defined in the **services** folder, allowing the Recommender Service to communicate asynchronously while providing the same behaviour as the endpoint by using the same Spring Boot components.

To get a recommendation from the Recommender Service, either by using the API or Kafka messaging, it is necessary to pass a user identifier as a parameter. Only by using a user identifier the Recommender Service can gather all the information it needs to provide a recommendation. Depending on the type of communication used, this identifier can be passed as an HTTP parameter or within the body of a Kafka message, as shown in Code snippets 10 and 11, respectively.

```
@Slf4j
@RestController
@RequiredArgsConstructor
public class RecommendationController {

    private final RecommendationService recommendationService;

    @GetMapping("recommendation")
    public ResponseEntity<?> getRecommendation(@RequestParam String userId) {
        try {
            String recommendationId =
recommendationService.getRecommendation(userId);
            return ResponseEntity.ok(new
RecommenderServiceResponseDto(recommendationId, userId));
        } catch (NoRecommendationFoundException e) {
            log.error(e.getMessage());
            return ResponseEntity.status(HttpStatus.NOT_FOUND).body(e.getMessage());
        } catch (Exception e) {
            return
ResponseEntity.status(HttpStatus.BAD_REQUEST).body(e.getMessage());
        }
    }
}
```

Code snippet 10 – RecommendationController.java

In Code snippet 10, more specifically in the **getRecommendation** method, it is possible to verify that it receives an HTTP request parameter called *userId* that represents the value of a user identifier.

```
@Slf4j
@Service
@RequiredArgsConstructor
public class ConsumerService {

    private final RecommendationService recommendationService;
    private final ProducerService producerService;

    @KafkaListener(topics = "recommender-service-topic", groupId = "recommender-
service-consumer-group")
    public void listen(@Header(KafkaHeaders.RECEIVED_TOPIC) String topicName,
@Payload String userId) {
        log.info("Received userId in topic {}: {}", topicName, userId);

        try {
            String recommendationId =
recommendationService.getRecommendation(userId);
            RecommenderServiceResponseDto recommenderServiceResponseDto = new
RecommenderServiceResponseDto(recommendationId, userId);
            producerService.sendMessage(recommenderServiceResponseDto);
        } catch (Exception e) {
            log.error(e.getMessage());
        }
    }
}
```

Code snippet 11 – ConsumerService.java

Similarly, in Code snippet 11, it is possible to see that the method **listen** receives a parameter called *userId* which also represents the value of a user identifier.

By further analysing both Code snippets 10 and 11, it is evident that both *RecommendationController* and *ConsumerService* proceed to handle the user identifier in the same way, using the *RecommendationService*, with the objective of generating a new recommendation.

RecommendationService.java is a Spring Boot component that was added to the Recommender Service that acts as layer between the application logic and the recommendation logic, while providing some exception handling, as shown by Code snippet 12.

```java
@Slf4j
@Service
@RequiredArgsConstructor
public class RecommendationService {

    private final SparkService sparkService;

    private final JdbcTemplate jdbcTemplate;

    public String getRecommendation(String userId) throws Exception {
        try {
            return sparkService.getRecommendation(userId);
        } catch (NoRecommendationPropertiesFoundException |
NoUserPropertiesFoundException e) {
            log.error(e.getMessage());
            log.info("Fetching random recommendation from database");
            String recId = getRandomUnrecommendedRecommendation(userId);
            return StringUtils.isNotEmpty(recId) ? recId :
getRandomRecommendationFromUserRecommendationHistory(userId);
        } catch (LackOfNewRecommendationsException e) {
            log.info(e.getMessage());
            log.info("Fetching random recommendation from recommendation_history
table where rating is lacking or is greater than 2");
            return getRandomRecommendationFromUserRecommendationHistory(userId);
        }
    }

    private String getRandomUnrecommendedRecommendation(String userId) {
        return jdbcTemplate.queryForObject(Constants.RANDOM_UNRECOMMENDED_REC_QUERY,
String.class, userId);
    }

    private String getRandomRecommendationFromUserRecommendationHistory(String
userId) {
        return
jdbcTemplate.queryForObject(Constants.RANDOM_REC_FROM_USER_REC_HISTORY_TABLE_QUERY,
String.class, userId, userId);
    }
}
```

Code snippet 12 - RecommendationService.java

This class contains three different methods:

- getRecommendation

This is the driver method. In conjunction with the next two methods, and by relying on *SparkService*, this method is capable of providing recommendations even if *SparkService* fails to do so, through the usage of checked exceptions.

- getRandomUnrecommendedRecommendation

When *SparkService* throws either a *NoRecommendationPropertiesFoundException* or a *NoUserPropertiesFoundException*, this method is responsible for connecting to the data source holding all the information related to the solution and getting a recommendation at random that has not yet been recommended to the user in the past.

- getRandomRecommendationFromUserRecommendationHistory

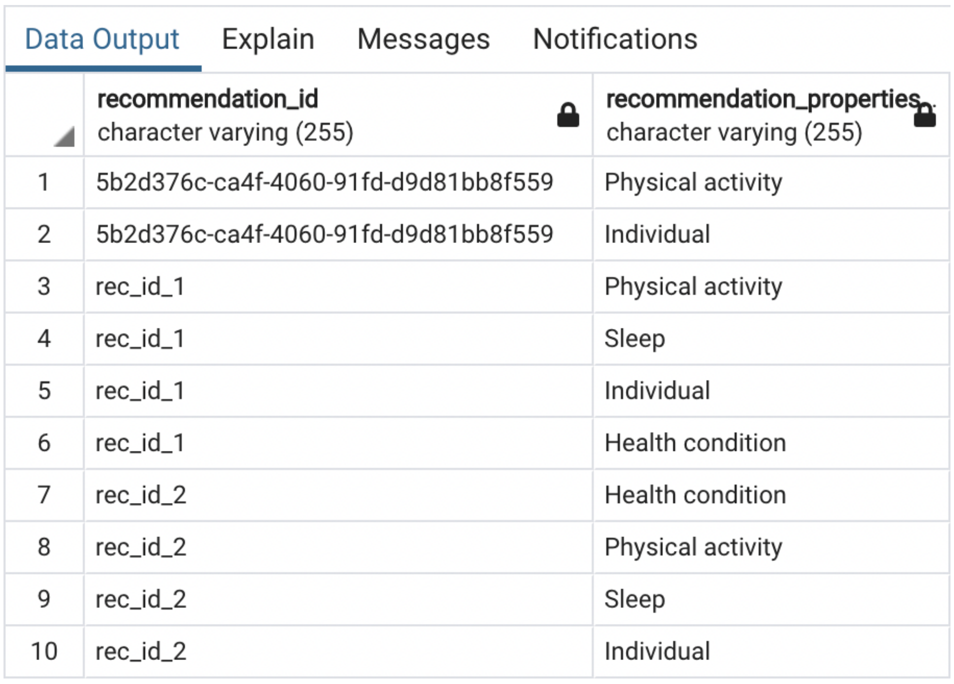
Similarly, when *SparkService* throws *LackOfNewRecommendationsException*, this method also connects to the data source holding all the information related to the solution and gets a recommendation at random that either was rated positively or that lacks a rating, meaning it might still be potentially interesting for the user receiving it.

SparkService.java is the Spring Boot component where Apache Spark library APIs are used, allowing for the calculation of similarities between recommendations and users' interests, and it is what powers the Recommender Service regarding its objective.

Generating a new recommendation can be enumerated as follows:

1. Build dataset with all identifiers and properties of recommendations not yet provided to the user, along with the identifier and properties of the user to whom the recommendation is for.
2. Group dataset by identifiers to obtain a dataset with a group of properties per identifier.
3. Vectorize the properties per identifier.
4. Calculate similarities between vectors, resulting in a data structure containing a combination of similarities between identifiers, per identifier.
5. Get the combination of similarities between the user identifier and all the recommendations identifiers.
6. Order the combination of similarities by similarity degree in descending order.
7. Get the first element of the combination of similarities.

To build the initial dataset, *SparkService* runs a query in the database that gets all the identifiers of recommendations not yet provided to the user in the solution, and each corresponding property. Considering the goal is to understand which recommendation is the most similar to the users' interests, this query also gets the user properties and places them, along with the user identifier, on the query result, as shown in Figure 48.



	recommendation_id character varying (255)	recommendation_properties character varying (255)
1	5b2d376c-ca4f-4060-91fd-d9d81bb8f559	Physical activity
2	5b2d376c-ca4f-4060-91fd-d9d81bb8f559	Individual
3	rec_id_1	Physical activity
4	rec_id_1	Sleep
5	rec_id_1	Individual
6	rec_id_1	Health condition
7	rec_id_2	Health condition
8	rec_id_2	Physical activity
9	rec_id_2	Sleep
10	rec_id_2	Individual

Figure 48 – Initial dataset query subset

However, this query may not return any data. This situation can happen for multiple reasons:

- The database does not have any recommendation.
- All recommendations have been provided to the user in the past.
- The database does not contain any recommendation with associated properties.
- The user for which the recommendation is for does not have any associated properties in the database.

If any of these situations occur, *SparkService* returns a checked exception back to *RecommendationService* so that the Recommender Service can return the most appropriate response.

Both the dataset initialization and the exception handling can be shown in Code snippet 13, and Figure 49 shows what the first twenty rows of the initial dataset look like, once initialized.

```
public String getRecommendation(String userId) throws Exception {

    Log.info("Getting recommendation for User with ID {}", userId);

    // get initial dataset with recommendation ids and properties + user id and
properties
    Dataset<Row> recDataset = getRecommendationsDatasetFromDB(userId);

    if (recDataset.isEmpty()) {
        // check database for possible reasons why initial dataset is empty

        Integer nrOfRecsInDB =
jdbcTemplate.queryForObject(Constants.CHECK_RECS_EXIST_QUERY, Integer.class);
        if (nrOfRecsInDB == null || nrOfRecsInDB == 0)
            throw new NoRecommendationFoundException("No recommendations found in the
database");

        Boolean userHasUnrecommendedItemsInDB =
jdbcTemplate.queryForObject(Constants.CHECK_USER_STILL_HAS_UNRECOMMENDED_RECS_QUERY,
Boolean.class, userId);
        if (userHasUnrecommendedItemsInDB == null || !userHasUnrecommendedItemsInDB)
{
            throw new LackOfNewRecommendationsException("There are no more
recommendations in the database that have not been recommended to User with ID" +
userId);
        }

        Integer nrOfRecPropsInDB =
jdbcTemplate.queryForObject(Constants.CHECK_RECS_PROPS_EXIST_QUERY, Integer.class);
        if (nrOfRecPropsInDB == null || nrOfRecPropsInDB == 0)
            throw new NoRecommendationPropertiesFoundException("No recommendation
properties found in the database");

        Integer nrOfUserPropsInDB =
jdbcTemplate.queryForObject(Constants.CHECK_USER_PROPS_EXIST_QUERY, Integer.class,
userId);
        if (nrOfUserPropsInDB == null || nrOfUserPropsInDB == 0)
            throw new NoUserPropertiesFoundException("No recommendation properties
found for user with ID " + userId + " in the database");
    }

    ...
}
```

Code snippet 13 - Dataset initialization and exception handling

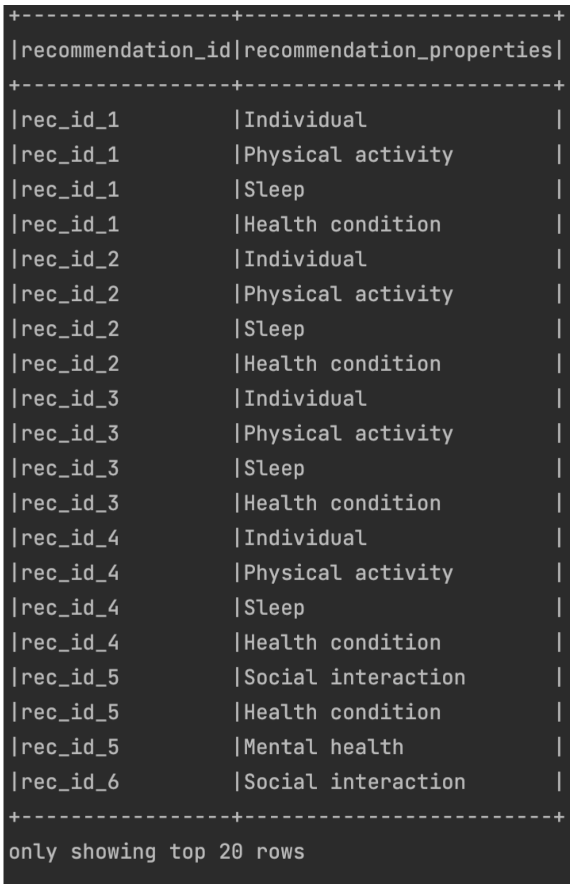
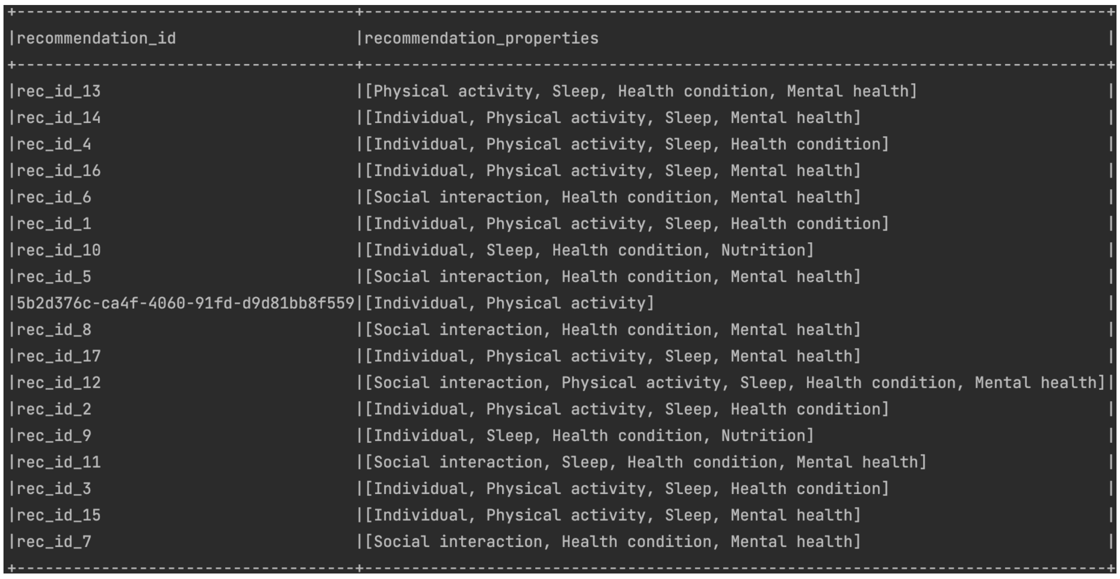

```
+----------------+------------------------+
|recommendation_id|recommendation_properties|
+----------------+------------------------+
|rec_id_1         |Individual              |
|rec_id_1         |Physical activity       |
|rec_id_1         |Sleep                   |
|rec_id_1         |Health condition        |
|rec_id_2         |Individual              |
|rec_id_2         |Physical activity       |
|rec_id_2         |Sleep                   |
|rec_id_2         |Health condition        |
|rec_id_3         |Individual              |
|rec_id_3         |Physical activity       |
|rec_id_3         |Sleep                   |
|rec_id_3         |Health condition        |
|rec_id_4         |Individual              |
|rec_id_4         |Physical activity       |
|rec_id_4         |Sleep                   |
|rec_id_4         |Health condition        |
|rec_id_5         |Social interaction      |
|rec_id_5         |Health condition        |
|rec_id_5         |Mental health           |
|rec_id_6         |Social interaction      |
+----------------+------------------------+
only showing top 20 rows
```

Figure 49 – First twenty rows of the initial dataset

Then, assuming the dataset was initialized with data, it is necessary to group the properties by identifier so that they can be vectorized, as shown in Code snippet 14 and demonstrated in Figure 50.

```java
public String getRecommendation(String userId) throws Exception {

    ...

    // group and aggregate initial dataset so that it can be worked on
    recDataset = recDataset.groupBy("recommendation_id").agg(new HashMap<>() {{
        put("recommendation_properties", "collect_list");
    }}).withColumnRenamed("collect_list(recommendation_properties)",
"recommendation_properties");

    ...
}
```

Code snippet 14 - Properties aggregation by identifier



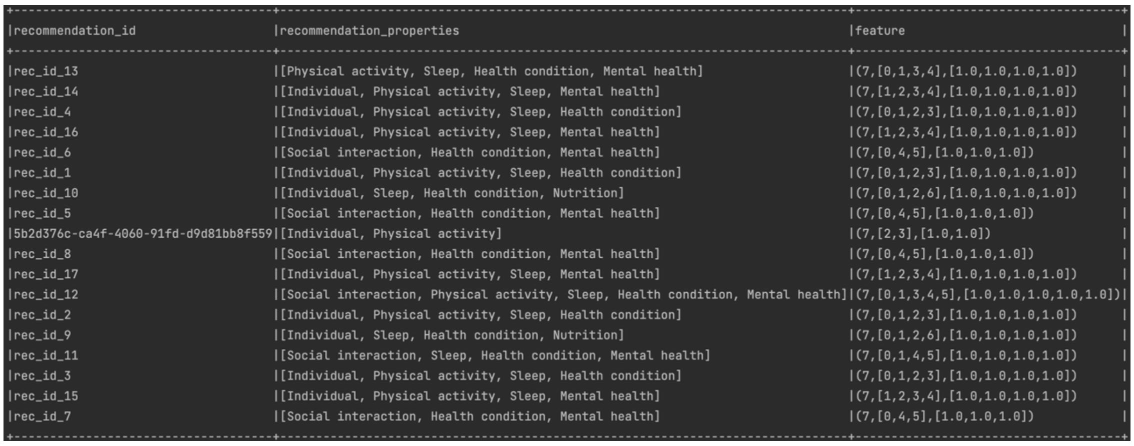
```
+------------------------------------+-------------------------------------------------------------------------------+
|recommendation_id                   |recommendation_properties                                                      |
+------------------------------------+-------------------------------------------------------------------------------+
|rec_id_13                           |[Physical activity, Sleep, Health condition, Mental health]                    |
|rec_id_14                           |[Individual, Physical activity, Sleep, Mental health]                          |
|rec_id_4                            |[Individual, Physical activity, Sleep, Health condition]                       |
|rec_id_16                           |[Individual, Physical activity, Sleep, Mental health]                          |
|rec_id_6                            |[Social interaction, Health condition, Mental health]                          |
|rec_id_1                            |[Individual, Physical activity, Sleep, Health condition]                       |
|rec_id_10                           |[Individual, Sleep, Health condition, Nutrition]                               |
|rec_id_5                            |[Social interaction, Health condition, Mental health]                          |
|5b2d376c-ca4f-4060-91fd-d9d81bb8f559|[Individual, Physical activity]                                                |
|rec_id_8                            |[Social interaction, Health condition, Mental health]                          |
|rec_id_17                           |[Individual, Physical activity, Sleep, Mental health]                          |
|rec_id_12                           |[Social interaction, Physical activity, Sleep, Health condition, Mental health]|
|rec_id_2                            |[Individual, Physical activity, Sleep, Health condition]                       |
|rec_id_9                            |[Individual, Sleep, Health condition, Nutrition]                               |
|rec_id_11                           |[Social interaction, Sleep, Health condition, Mental health]                   |
|rec_id_3                            |[Individual, Physical activity, Sleep, Health condition]                       |
|rec_id_15                           |[Individual, Physical activity, Sleep, Mental health]                          |
|rec_id_7                            |[Social interaction, Health condition, Mental health]                          |
+------------------------------------+-------------------------------------------------------------------------------+
```

Figure 50 - Aggregated dataset

After grouping the properties by identifier, it is used the *CountVectorizerModel* from the Apache Spark library, to create a vector from each group of properties, as shown in Code snippet 15 and Figure 51.

```
public String getRecommendation(String userId) throws Exception {

    ...

    // use CountVectorizer to create a vector out of recommendation properties for
each recommendation
    CountVectorizerModel recCountVecModel = new CountVectorizer()
            .setInputCol("recommendation_properties")
            .setOutputCol("feature")
            .fit(recDataset);

    Dataset<Row> recDatasetVectorized = recCountVecModel.transform(recDataset);

    ...
}
```

Code snippet 15 - Properties vectorization



```
+------------------------------------+-------------------------------------------------------------------------------+-------------------------------------+
|recommendation_id                   |recommendation_properties                                                      |feature                              |
+------------------------------------+-------------------------------------------------------------------------------+-------------------------------------+
|rec_id_13                           |[Physical activity, Sleep, Health condition, Mental health]                    |(7,[0,1,3,4],[1.0,1.0,1.0,1.0])      |
|rec_id_14                           |[Individual, Physical activity, Sleep, Mental health]                          |(7,[1,2,3,4],[1.0,1.0,1.0,1.0])      |
|rec_id_4                            |[Individual, Physical activity, Sleep, Health condition]                       |(7,[0,1,2,3],[1.0,1.0,1.0,1.0])      |
|rec_id_16                           |[Individual, Physical activity, Sleep, Mental health]                          |(7,[1,2,3,4],[1.0,1.0,1.0,1.0])      |
|rec_id_6                            |[Social interaction, Health condition, Mental health]                          |(7,[0,4,5],[1.0,1.0,1.0])            |
|rec_id_1                            |[Individual, Physical activity, Sleep, Health condition]                       |(7,[0,1,2,3],[1.0,1.0,1.0,1.0])      |
|rec_id_10                           |[Individual, Sleep, Health condition, Nutrition]                               |(7,[0,1,2,6],[1.0,1.0,1.0,1.0])      |
|rec_id_5                            |[Social interaction, Health condition, Mental health]                          |(7,[0,4,5],[1.0,1.0,1.0])            |
|5b2d376c-ca4f-4060-91fd-d9d81bb8f559|[Individual, Physical activity]                                                |(7,[2,3],[1.0,1.0])                  |
|rec_id_8                            |[Social interaction, Health condition, Mental health]                          |(7,[0,4,5],[1.0,1.0,1.0])            |
|rec_id_17                           |[Individual, Physical activity, Sleep, Mental health]                          |(7,[1,2,3,4],[1.0,1.0,1.0,1.0])      |
|rec_id_12                           |[Social interaction, Physical activity, Sleep, Health condition, Mental health]|(7,[0,1,3,4,5],[1.0,1.0,1.0,1.0,1.0])|
|rec_id_2                            |[Individual, Physical activity, Sleep, Health condition]                       |(7,[0,1,2,3],[1.0,1.0,1.0,1.0])      |
|rec_id_9                            |[Individual, Sleep, Health condition, Nutrition]                               |(7,[0,1,2,6],[1.0,1.0,1.0,1.0])      |
|rec_id_11                           |[Social interaction, Sleep, Health condition, Mental health]                   |(7,[0,1,4,5],[1.0,1.0,1.0,1.0])      |
|rec_id_3                            |[Individual, Physical activity, Sleep, Health condition]                       |(7,[0,1,2,3],[1.0,1.0,1.0,1.0])      |
|rec_id_15                           |[Individual, Physical activity, Sleep, Mental health]                          |(7,[1,2,3,4],[1.0,1.0,1.0,1.0])      |
|rec_id_7                            |[Social interaction, Health condition, Mental health]                          |(7,[0,4,5],[1.0,1.0,1.0])            |
+------------------------------------+-------------------------------------------------------------------------------+-------------------------------------+
```

Figure 51 - Vectorized dataset

After vectorizing all properties, to calculate the similarity between vectors it is used the *columnSimilarities* method, also from Apache Spark. However, because this method calculates similarities between columns instead of rows, it was necessary to create a matrix and transpose it, as shown by Code snippet 16.

```java
public String getRecommendation(String userId) throws Exception {

    ...

    // calculate cosine similarity between each recommendation
    List<IndexedRow> indexedRows = new ArrayList<>();
    Map<Long, String> indexedRowsRecIdsMap = new HashMap<>();

    long i = 0;
    for (Row row : recDatasetVectorized.collectAsList()) {
        indexedRowsRecIdsMap.put(i, row.get(0).toString());
        Vector vector = Vectors.dense(((SparseVector) row.get(2)).toArray());
        indexedRows.add(new IndexedRow(i++, vector));
    }

    Dataset<IndexedRow> indexedRowDataset = sparkSession.createDataset(indexedRows,
Encoders.javaSerialization(IndexedRow.class));
    RDD<IndexedRow> indexedRowRDD = indexedRowDataset.rdd();

    IndexedRowMatrix indexedRowMatrix = new IndexedRowMatrix(indexedRowRDD);
    BlockMatrix blockMatrix = indexedRowMatrix.toBlockMatrix().transpose();
    IndexedRowMatrix transposedMatrix = blockMatrix.toIndexedRowMatrix();

    RDD<MatrixEntry> matrixEntryRDD =
transposedMatrix.columnSimilarities().entries();

    ...
}
```

Code snippet 16 - Similarities calculation

Because these operations rely on the Apache Spark library and its corresponding data structures, it is then created a map to facilitate getting the similarities between the user and recommendations. Finally, to get the ideal recommendation, the user-recommendations similarities are ordered by similarity degree in descending order and the recommendation with the highest degree in relation to the interests of the user is returned, as shown by Code snippets 17, 18, and 19, respectively.

```
public String getRecommendation(String userId) throws Exception {

    ...

    List<MatrixEntry> matrixEntries = matrixEntryRDD.toJavaRDD().collect();
    Map<String, Map<String, Double>> similaritiesMap = new HashMap<>();

    for (MatrixEntry entry : matrixEntries) {
        String recommendationIdFrom =
indexedRowsRecIdsMap.get(entry.copy$default$1()); // get source recommendation ID
        String recommendationIdTo = indexedRowsRecIdsMap.get(entry.copy$default$2());
// get destination recommendation ID
        Double cosineSimilarity = entry.copy$default$3(); // get cosine similarity
between source and destination recommendation ID

        Map<String, Double> similaritiesToDefault$1 =
similaritiesMap.computeIfAbsent(recommendationIdFrom, k -> new HashMap<>()); // if
similaritiesMap.get(recommendationIdFrom) exists then gets the value, if not then it
adds a new HashMap() as the value
        Map<String, Double> similaritiesToDefault$2 =
similaritiesMap.computeIfAbsent(recommendationIdTo, k -> new HashMap<>()); // if
similaritiesMap.get(recommendationIdTo) exists then gets the value, if not then it
adds a new HashMap() as the value

        similaritiesToDefault$1.put(recommendationIdTo, cosineSimilarity);
        similaritiesToDefault$2.put(recommendationIdFrom, cosineSimilarity);
    }

    ...
}
```

Code snippet 17 - Similarities map

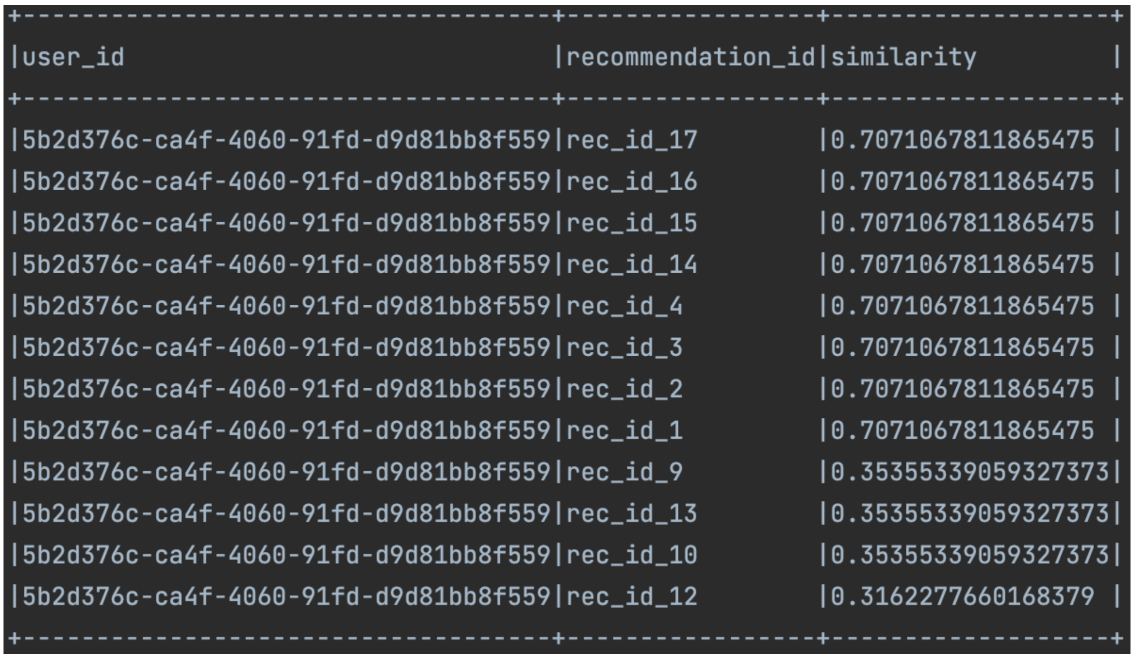
```
public String getRecommendation(String userId) throws Exception {

    ...

    // get cosine similarity matrix between user and all recommendations
    Log.info("Similarities matrix: {}", similaritiesMap);
    Map<String, Double> userRecSimilarities = similaritiesMap.get(userId);
    Log.info("User-Recommendation similarities: {}", userRecSimilarities);

    // order user cosine similarity matrix by descending order
    LinkedHashMap<String, Double> userRecSimilaritiesSortByValueDesc = new
LinkedHashMap<>();

userRecSimilarities.entrySet().stream().sorted(Map.Entry.comparingByValue(Comparator.
reverseOrder()))
            .forEachOrdered(x -> userRecSimilaritiesSortByValueDesc.put(x.getKey(),
x.getValue()));

    Log.info("User-Recommendation similarities in descending order: {}",
userRecSimilaritiesSortByValueDesc);

    ...
}
```

Code snippet 18 - Order similarities map in descending order by similarity degree

```
public String getRecommendation(String userId) throws Exception {

    ...

    String recommendationId = new
ArrayList<>(userRecSimilaritiesSortByValueDesc.entrySet()).get(0).getKey();

    return recommendationId;
}
```

Code snippet 19 - Get and return recommendation with highest similarity to interests of user

Using the information shown in Figures 49 to 51 as an example, the user-recommendations similarities ordered by similarity in descending order, for the same data, can be represented by Figure 52.



```
+------------------------------------+-----------------+-------------------+
|user_id                             |recommendation_id|similarity         |
+------------------------------------+-----------------+-------------------+
|5b2d376c-ca4f-4060-91fd-d9d81bb8f559|rec_id_17        |0.7071067811865475 |
|5b2d376c-ca4f-4060-91fd-d9d81bb8f559|rec_id_16        |0.7071067811865475 |
|5b2d376c-ca4f-4060-91fd-d9d81bb8f559|rec_id_15        |0.7071067811865475 |
|5b2d376c-ca4f-4060-91fd-d9d81bb8f559|rec_id_14        |0.7071067811865475 |
|5b2d376c-ca4f-4060-91fd-d9d81bb8f559|rec_id_4         |0.7071067811865475 |
|5b2d376c-ca4f-4060-91fd-d9d81bb8f559|rec_id_3         |0.7071067811865475 |
|5b2d376c-ca4f-4060-91fd-d9d81bb8f559|rec_id_2         |0.7071067811865475 |
|5b2d376c-ca4f-4060-91fd-d9d81bb8f559|rec_id_1         |0.7071067811865475 |
|5b2d376c-ca4f-4060-91fd-d9d81bb8f559|rec_id_9         |0.35355339059327373|
|5b2d376c-ca4f-4060-91fd-d9d81bb8f559|rec_id_13        |0.35355339059327373|
|5b2d376c-ca4f-4060-91fd-d9d81bb8f559|rec_id_10        |0.35355339059327373|
|5b2d376c-ca4f-4060-91fd-d9d81bb8f559|rec_id_12        |0.3162277660168379 |
+------------------------------------+-----------------+-------------------+
```

Figure 52 - Similarity between a user and recommendations

Considering the description on how the recommendation logic operates, in the scenario represented by Figure 52, it is possible to conclude that the recommendation with identifier *rec_id_17* is the eligible one for having one of the highest similarity metrics and being the first element on the data structure.

6 Evaluation and Experimentation

In this chapter, it is presented the evaluation of the implemented solution. For this, an hypothesis is defined, as well as the evaluation methodology, and metrics.

6.1 Hypotheses

With the goal of proving the implemented solution to be of quality and capable of providing adequate stress-easing recommendations, the following hypothesis is defined:

- The system provides adequate stress-easing recommendations to users.

In the context of this document, an adequate stress-easing recommendation is assumed to be a recommendation that has been positively evaluated by the user that received it.

6.2 Methodology

To prove that the system can provide adequate stress-easing recommendations, it is necessary to collect user-generated data. This data corresponds to the ratings from the users regarding the recommendations that were made in the past, on a scale from one to five, and it is possible to obtain through the access to the database where they are stored.

However, this solution is part of an H2020 initiative which involves other parties in the delivery process. For this reason, as of the date of this document, there is no user-generated data that can be used for the purpose of evaluation.

To counter the lack of user-generated data, it was used randomly generated data. For that reason, the results are merely demonstrative and are not subject to interpretation. It is only intended to show how the system can be evaluated.

Regarding the generated data, it was considered the following:

- 10 users
- 0 to 7 randomly generated recommendations per user
- Randomly generated scores from 1 to 5 to the generated recommendations per user

6.3 Metrics

An effective recommender system is expected to provide appropriate recommendations that users enjoy and at the same time avoid recommendations that users dislike. Thus, it is possible to assess the effectiveness of the system by measuring both error metrics and decision support metrics.

By measuring error metrics, it is possible to verify how much the recommender system deviates from users' interests, meaning the smaller these metrics, the better. This way, the error can be represented as the rating from a user r minus the rating predicted by the recommender system p for a given recommendation, such that:

$$error = |r - p| \tag{16}$$

By summing all the errors associated to each recommendation for a given user and dividing the result by the number of rated recommendations, we get the Mean Absolute Error (MAE):

$$MAE = \frac{\sum_{i=1}^{N} |r_i - p_i|}{N}, \tag{17}$$

where N is the number of recommendations

As of the date of this document, the recommender system has no concern for a specific rating when providing recommendations, meaning its purpose is to generate the best suitable recommendation based on the information it detains, such as users' preferences. However, it is possible to perform a more rigorous assessment by assuming the maximum rating of 5 for all the recommendations provided by the system. This way, the ratings from the users will be compared to the rating of 5.

On the other hand, measuring decision support metrics consists of evaluating each recommendation as right or wrong, and comparing the system recommendations to users' decisions, it is possible to consider the following outcomes:

True Positive (TP) – The recommendation is provided by the system and enjoyed by the user.

False Positive (FP) – The recommendation is provided by the system, but the user disliked it.

True Negative (TN) – The system did not provide a recommendation that the user would have enjoyed.

False Negative (FN) – The system did not provide a recommendation, but the user would not have enjoyed it anyway.

Recommendations are not available to users until they are provided by the system, meaning that users will not have access to them otherwise, which led to the exclusion of both TN and FN outcomes from this evaluation, as shown in Table 13.

Table 13 - Confusion Matrix

		System Recommendation	
		Proposed	Not Proposed
User	Liked	TP	FN
	Disliked	FP	TN

From this confusion matrix, it is only possible to define the precision metric to evaluate the recommender system, which gives the fraction of the recommendations that users liked. However, this not a problem since it is often considered more important to optimize precision, while not putting too much importance on whether users get all the possible relevant recommendations, which is represented by the recall metric. Precision can be calculated as follows:

$$Precision = \frac{TP}{TP + FP} \tag{18}$$

Considering that user ratings to recommendations provided by the system can vary from 1 to 5, it is assumed that ratings from 1 to 2 are represented as FP and ratings from 3 to 5 are represented as TP.

6.4 System Evaluation

By applying the methodology described in subchapter 6.2, and considering the metrics defined in subchapter 6.3, it is possible to obtain the results shown in Tables 14 and 15.

Table 14 - MAE calculation

user	numRec	score1	score2	score3	score4	score5	score6	score7	MAE
user1	4	2	4	1	5	0	0	0	2
user2	3	2	2	4	0	0	0	0	2,33
user3	4	1	1	5	4	0	0	0	2,25
user4	4	5	3	2	2	0	0	0	2
user5	6	3	2	2	2	3	1	0	2,83
user6	3	3	3	1	0	0	0	0	2,67
user7	4	1	1	3	1	0	0	0	3,5
user8	3	1	2	5	0	0	0	0	2,33
user9	6	1	1	1	3	3	1	0	3,33
user10	6	4	4	2	1	4	5	0	1,67

Table 15 - Precision calculation

user	numRec	score1	score2	score3	score4	score5	score6	score7	TP	FP	Precision
user1	4	2	4	1	5	0	0	0	2	2	0,5
user2	3	2	2	4	0	0	0	0	1	2	0,33
user3	4	1	1	5	4	0	0	0	2	2	0,5
user4	4	5	3	2	2	0	0	0	2	2	0,5
user5	6	3	2	2	2	3	1	0	2	4	0,33
user6	3	3	3	1	0	0	0	0	2	1	0,67
user7	4	1	1	3	1	0	0	0	1	3	0,25
user8	3	1	2	5	0	0	0	0	1	2	0,33
user9	6	1	1	1	3	3	1	0	2	4	0,33
user10	6	4	4	2	1	4	5	0	4	2	0,67

These metrics permit to do an individual evaluation, specific to a user, or a global evaluation of the system, and can be used as a guidance for future system improvements.

7 Conclusion

This document presents the study and the development of a recommender system within the context of an application related to stress, with the intent of aiding its users to deal with episodes of stress and of increasing awareness regarding the side effects involved.

Prior to the conducted study leading to the implementation that was documented, several meetings took place with experts in the area to better understand the purpose of the project and to collect all the necessary information related to the theme, as well as how a recommender system could address the matter, allowing for the definition of the problem and the objective. It was then defined that a web application would be a good option to support the interaction between the target audience and the recommender system.

Afterwards, it was conducted a state of the art about recommender systems within the context of stress and the types of filtering techniques that are commonly used, including their advantages and disadvantages. Considering that the implemented solution might have a small number of users throughout the first iterations, the content-based filtering technique was chosen as the main filtering technique for the solution, allowing for recommendations to be provided without the concern for sparse data.

Furthermore, a value analysis was realized, describing the perceived value of the proposed solution, with the aim of exposing the advantages that would justify its development from the perspective of the client.

Considering the requirements that were defined during the meetings with the client, an architectural proposal was created containing two alternatives, where its components are detailed, as well as the reason behind the selection of one over the other.

The purpose of this project stands on the possibility of providing an engaging application for users to interact with, to diminish and eventually avoid the impacts of stress on the day-to-day basis through adequate recommendations. For this, it was necessary to define the following goals:

- Build a web application that is visually appealing and interactive

  Visual appeal is subjective, and what some might perceive as eye catching, others may feel indifferent. However, as presented in subchapter 5.1, the graphical user interface is consistent between every page and component within them. The color palette blends in very well and transmits a soft experience due to the rounded and fading interface components.

  Furthermore, it highlights clickable components such as buttons and list elements, and has a notification animation, making it easier to spot the interactive aspects of the application.

- Build a recommender system capable of providing health-related recommendations

  After deciding which filtering technique to use, it was necessary to apply the logic behind the technique in a way such that it integrates with the overall solution. As presented and described in subchapter 5.5, although some data preparation was necessary, a functioning recommender system was achieved with the aid of Apache Spark, an open-source Java library known for providing machine learning algorithms and mathematical computation, such as vectorization and calculation of cosine similarities, both crucial for a content-based recommender system to work.

- Integrate the recommender system in the web application

  With a functioning web application and a recommender system, getting both components to communicate was the final piece to the main goal of this solution. As of the date of this document there are not many recommendations in the system. However, that number is expected to grow in the future, meaning that the amount of time required to calculate new recommendations will certainly increase as well. Having adopted a microservices-oriented approach, to prevent a bottleneck between the web application and the recommender system, this communication was established by leveraging Kafka capabilities.

  To further enrich the solution, it was also integrated a rules engine in the application, allowing for an important aspect of it, the questionnaire, to be easily maintainable and customizable.

Although all defined goals were achieved, there is opportunity for future work, such as:

- Deploy the solution in a production environment
- Adapt the solution for mobile usage
- Include geolocation
- Take geolocation into account when providing recommendations
- Include a back-office web page for admins of the application to configure recommendations, properties, questions, and other relevant aspects that are currently setup through database queries
- Include a dashboard web page with health-related graphs and define it as the homepage for the application

References

Bouraga, S., Jureta, I., Faulkner, S., and Herssens, C., 2014. Knowledge-Based Recommendation Systems
https://doi.org/10.4018/ijiit.2014040101

Burke, R., 1999. Integrating Knowledge-based and Collaborative-filtering Recommender Systems. https://www.aaai.org/Papers/Workshops/1999/WS-99-01/WS99-01-011.pdf (accessed on 19/01/2022)

Chung, L., Nixon, B., A., Yu, E., & Mylopoulos, J., 2000. Non-Functional Requirements in Software Engineering. https://doi.org/10.1007/978-1-4615-5269-7

Clarke, S., Jaimes, L., G., & Labrador, M., A., 2017. mStress: a Mobile Recommender System for Just-in-Time Interventions for Stress. https://doi.org/10.1109/CCNC.2017.8015367

Croon, R. De, Houdt, L., V., Htun, N., N., Stiglic, G., Abeele, V., V., & Verbert, K., 2021. Health Recommender Systems: Systematic Review. https://doi.org/10.2196/18035

Fu, G., Zhang, Y., & Yu, G., 2020. A Fair Comparison of Message Queuing Systems. https://doi.org/10.1109/ACCESS.2020.3046503

Gupta, J., & Gadge, J., 2014. A Framework for a Recommendation System Based on Collaborative Filtering and Demographics. https://doi.org/10.1109/CSCITA.2014.6839276

Hiremath, B., 2016. An Alteration of the Web 1.0, Web 2.0 and Web 3.0: A Comparative Study. https://doi.org/10.13140/2.1.2287.2961

Kaur, H., & Bathla, G., 2019. Techniques of Recommender System. https://doi.org/10.35940/ijitee.I1059.0789S19

Kuanr, M., & Mohapatra, P., 2021. Assessment Methods for Evaluation of Recommender Systems: A Survey. https://doi.org/10.2478/fcds-2021-0023

Knaster, R., 2021. Advanced topic – domain modeling. https://www.scaledagileframework.com/domain-modeling/ (accessed on 12/04/2022)

Lerato, M., Esan, O., A., Ebunoluwa, A., Ngwira, S., M., & Zuva, T., 2016. A Survey of Recommender System Feedback Techniques, Comparison and Evaluation Metrics. https://doi.org/10.1109/CCCS.2015.7374146

Li, L., Hsu, R., & Lee, F., 2011. Review of Recommender Systems and Their Applications. https://www.semanticscholar.org/paper/Review-of-Recommender-Systems-and-Their-Application-Li-Hsu/5f4dc82c24ad41e4d246a91bacbeeb38cf70ee50 (accessed on 08/01/2022)

Lu, L., Medo, M., Yeung, C., H., Zhang, Y., Zhang, Z., & Zhou, T., 2012. Recommender systems. https://doi.org/doi:10.1016/j.physrep.2012.02.006

Meteren, R. van, 2000. Using Content-Based Filtering for Recommendation. https://www.semanticscholar.org/paper/Using-Content-Based-Filtering-for-Recommendation-Meteren/4a57e0f0641b7a70fece89c14fbf5030869ededb (accessed on 15/01/2022)

Mustafa, N., Ibrahim, A., O., Ahmed, A., & Abdullah, A., 2017. Collaborative Filtering: Techniques and Applications. https://doi.org/10.1109/ICCCCEE.2017.7867668

Paredes, P., E., Gilad-Bachrach, R., Czerwinski, M., & Roseway, A., 2014. PopTherapy: Coping with Stress through Pop-Culture. https://doi.org/10.4108/icst.pervasivehealth.2014.255070

Panigrahi, S., Lenka, R., K., & Stitipragyan, A., 2016. A Hybrid Distributed Collaborative Filtering Recommender Engine Using Apache Spark. https://doi.org/10.1016/j.procs.2016.04.214

Rosa, R., Schwartz, G., M., Ruggiero, W., V., & Rodriguez, D., Z., 2018. A Knowledge-Based Recommendation System That Includes Sentiment Analysis and Deep Learning. https://doi.org/10.1109/TII.2018.2867174

Rosen, M., A., & Boccia, F., 2021. Is Consumer Overchoice a Reason for Decision Paralysis? https://doi.org/10.3390/su13115920

Shah, K., & Salunke, A., 2017. Recommender Systems: An overview of different approaches to recommendations. https://doi.org/10.1109/ICIIECS.2017.8276172

Sharma, R., & Singh, R., K., 2016. Evolution of Recommender Systems from Ancient Times to Modern Era: A Survey. https://doi.org/10.17485/ijst/2016/v9i20/88005

Shin, I., Cha, J., Cheon, G., W., Lee, C., Lee, S., Y., Yoon, H., & Kim, H., C., 2014. Automatic stress-relieving music recommendation system based on photoplethysmography-derived heart rate variability analysis. https://doi.org/10.1109/EMBC.2014.6945093

Vlezen, C., V., 2018. Business Rules Management Systems. https://studenttheses.uu.nl/bitstream/handle/20.500.12932/29582/brms.pdf?sequence=2 (accessed on 10/08/2022)

Vohra., D., 2016. Apache Kafka. https://doi.org/10.1007/978-1-4842-2199-0_9

Walunj, S., & Sadafale, K., 2013. An Online Recommendation System for E-commerce Based on Apache Mahout Framework. https://doi.org/10.1145/2487294.2487328

Xin, Y., Chen, Y., Jin, L., Cai, Y., & Feng, L., 2017. TeenRead: An Adolescents Reading Recommendation System Towards Online Bibliotherapy. https://doi.org/10.1109/BigDataCongress.2017.63