



---

**TECHNICAL UNIVERSITY**

OF CLUJ-NAPOCA, ROMANIA

**FACULTY OF AUTOMATION AND COMPUTER SCIENCE  
COMPUTER SCIENCE DEPARTMENT**

# **FOOD RECOMMENDER SYSTEM**

LICENSE THESIS

Graduate: **Robert-Stefan DOMOKOS**

Supervisor: **Prof. dr. eng. Ionut ANGHEL**

**2023**

## Table of contents

<b>Chapter 1. Introduction.....</b>	<b>1</b>
<b>Chapter 2. Project Objectives.....</b>	<b>3</b>
2.1.    Main Objectives.....	3
2.2.    Secondary Objectives.....	3
<b>Chapter 3. Bibliographic Research.....</b>	<b>6</b>
3.1.    Personalized nutrition.....	6
3.1.1.  Body Mass Index (BMI) and Basal Metabolic Rate (BMR).....	6
3.1.2.  Nutritional strategies and dataset.....	7
3.2.    Recommender systems.....	8
3.3.    Multi-criteria decision analysis (MCDA).....	8
3.4.    Machine Learning (ML).....	9
3.4.1.  Introduction.....	9
3.4.2.  Machine Learning in healthcare and nutrition.....	11
3.5.    Related works.....	11
3.5.1.  DIETOS: a recommender system for health profiling and diet management .....	12
3.5.2.  A cross-domain framework for designing healthcare mobile applications .....	12
3.5.3.  SousChef: Mobile Meal Recommender System for Older Adults....	13
<b>Chapter 4. Analysis and Theoretical Foundation.....</b>	<b>14</b>
4.1.    The Description of the Dataset.....	14
4.2.    Analytic Hierarchy Process (AHP).....	15
4.2.1.  The Principles and Axioms of AHP.....	16
4.2.2.  The Phases of AHP.....	17
4.3.    K-Nearest Neighbors (KNN).....	18
4.3.1.  Introduction to KNN.....	18
4.3.2.  Mathematical Background of the KNN.....	19
4.3.3.  Feature Transformation Techniques.....	20
4.3.4.  The flow of the KNN algorithm.....	20
4.4.    Evaluation Metrics.....	21
4.4.1.  Acuracy.....	21
4.4.2.  Confusion Matrix.....	22

<b>Chapter 5. Detailed Design and Implementation.....</b>	<b>24</b>
5.1. Detailed Design.....	24
5.1.1. Main Architecture and Use Cases.....	24
5.1.2. Backend and Database Design.....	29
5.1.3. Used Technologies and Frameworks.....	31
5.2. Implementation.....	34
5.2.1. The Recommender System.....	34
5.2.2. Backend App.....	41
5.2.3. React Frontend App.....	47
<b>Chapter 6. Testing and Validation.....</b>	<b>51</b>
<b>Chapter 7. User's Manual.....</b>	<b>54</b>
7.1. Installation process.....	54
7.1.1. Backend and Database connection setup.....	54
7.1.2. Frontend setup.....	55
7.1.3. Recommender system app.....	55
7.2. How to use our system.....	56
7.2.1. Login and Register.....	56
7.2.2. Client's manual.....	57
7.2.3. Nutritionist's manual.....	59
<b>Chapter 8. Conclusions.....</b>	<b>61</b>
<b>Bibliography.....</b>	<b>63</b>

## Chapter 1. Introduction

In the modern era, the importance of maintaining optimal nutrition and its profound impact on overall health and well-being cannot be emphasized enough. As our lives become highly fast-paced and significantly bound with the latest technology, our dietary habits have experienced a significant change, resulting in unexpected consequences for our health. The current scenario creates a difficult paradox that lies in the large array of food alternatives, but lack of comprehension and awareness of the nutritional content and its implications for human health.

Consistently making unwise food choices can have serious and far-reaching repercussions. Current eating habits that are often heavy in processed foods, sugars, unhealthy fats, and empty calories are causing a large array of harmful effects on health. One of the most known and alarming effects is the rise in cases of obesity, which not only impacts how individuals look but additionally dramatically increases the risk in serious medical conditions like heart diseases, stroke, or particular types of cancer. Furthermore, poor nutrition may lead to nutritional deficiencies which can limit the body's ability to perform at its optimal level. For instance, bad fats and excessive sugar could affect the neurotransmitter system and produce depression and anxiety.

On the other hand, a balanced meal plan forms a crucial foundation for supporting longevity and good health. The core requirement for a strong immune system, a healthy metabolism, and long-lasting energy comes from nutrient-rich foods. Proteins, carbohydrates, healthy fats, vitamins, and minerals are vital for repairing and maintaining body tissues, supporting cognitive function, and avoiding chronic illnesses.

The process of designing a nutritious and healthy meal plan is a challenging puzzle to solve. While the notion of a healthy diet is universal, the concept of "healthy" differs from person to person. Everyone has unique dietary requirements that depend on multiple attributes and factors such as age, height, weight, physical activity level and lifestyle objectives, but also each individual has their own culinary palette of preferences. The meal plan must take into consideration not only the nutritional needs and objectives that a certain person has, but also must include the person's particular preferences. Incorporating individual preferences into meal planning is a critical strategy for increasing engagement and adherence to the recommended plan, making the process of following the diet more pleasurable and fulfilling.

The way we tackle the challenge of creating such meal plans that can satisfy both the user's needs and preferences has completely changed because of the convergence of the latest innovations in recommender systems and machine learning. Recommender systems, which are frequently seen in platforms dedicated for streaming services or in e-commerce, are using certain algorithms to analyze user preferences and behavior to offer new personalized recommendations. These systems have the ability to dynamically design meal plans that match a person's nutritional needs, constraints and also their desires.

Along with these systems and a comprehensive database of nutritional information of a vast number of edible products, machine learning technologies are needed to analyze these database resources in order to design meal plans that satisfy particular preferences and meet specific health objectives. By taking advantage of machine learning algorithms, the process of discovering suitable food combinations and amounts becomes far more efficient and accurate than traditional manual methods.

Given the complexity of creating an efficient and healthy menu plan, that also focusses on integrating user's preferences, I designed the Food Recommender System to provide a dynamic and personalized solution to this matter. The application fills the gap between generalized nutritional guidance and the specifics of individual dietary by using multiple technologies like recommender systems and machine learning. This application is presented as a tool for not only users who want to create personalized meal plans, but also nutritionists to help them keep track of their client's nutritional needs and also prescribe new plans.

## Chapter 2. Project Objectives

As presented previously in the first chapter, the primary aim of this license thesis project revolves around the development and design of a food recommender system that will not only be a tool for the users that want to change or improve their nutrition and diet but also a tool for nutritionists to keep track of the meal plans that their clients have or prescribe new ones. That being said, the system's main and secondary objectives must be clearly specified.

### 2.1. Main Objectives

As mentioned before, the main focus in the development of this recommender system lies in the generation of personalized meal plans that are meeting each user's unique needs from a dietary and sensory perspective. The major goal can be further broken down into two key components: satisfying the necessary nutritional requirements that are particular to each user and smoothly incorporating their personal preferences into the resulting meal plan. Both components are profoundly bound to the need of a vast and comprehensive dataset that mainly focusses on the nutritional composition of edible products.

The first aspect concentrates on the robust examination of the food entries, that are available in the dataset, to discard those that do not correspond to the specific user's dietary restrictions and health objectives. In the real world, this process is carefully executed by a nutritionist, who has access to his client's nutritional information, but in our application, we make use of a multi criteria decision analysis algorithm that can make this operation much faster. The second component centers on the seamless integration of personal preferences, process that our system accomplishes with the help of machine learning algorithms, that are assigned with the recommendation of new food items that will correspond with the given preferences.

### 2.2. Secondary Objectives

For achieving the main objectives of this license thesis, we must elaborate and carefully define the secondary objectives, objectives that not only strengthen the system's foundation, but also form a path to a clearer understanding of the main goal:

- *Dataset Exploration and Selection:* Careful investigation and selection of the ideal dataset is a crucial objective. This implies a thorough examination of the nutritional databases that are accessible in order to select one that is relevant, reliable, robust and encapsulates a wide array of consumables. The effectiveness of our system relies significantly on the completeness of the chosen dataset. I found that the USDA National Nutrient Database for Standard Reference is suitable for this task. This particular dataset provides around 7,800 distinct food items each accompanied by their nutritional compositions measured per 100 grams.
- *Multi-Criteria Decision Analysis (MCDA) Framework:* Finding and implementing a successful MCDA strategy is a significant objective in order to optimize the selection of appropriate food items. This framework acts as a tactical tool to impartially assess and classify meals based on specified criteria, in our case the macronutrients and micronutrients of the food entries such as the number of proteins, carbohydrates, fats and many more. Due to its ability to manage

complicated decision-making scenarios, the Analytic Hierarchy Process (AHP) algorithm comes as the best candidate for our case, being a strong option for optimizing the selection procedure.

- *Machine Learning Algorithm Selection:* Another crucial objective revolves on identifying the most appropriate ML algorithm for the integration of food preferences used for the generation of new food recommendations. For a recommender system this task involves rigorous analysis of various algorithms, taking into account their precision, accuracy and efficiency. In our case, an advantageous choice is the K-Nearest Neighbors (KNN) algorithm.
- *User's Nutritional Data Gathering:* A vital step in implementing this recommender system is the user's nutritional data gathering. The system must collect the user's nutritional information, such as age, height, weight, health conditions, weight goals, activity levels and so on in order to perfectly set the user's unique dietary needs. This is achieved through the implementation of a user-friendly and intuitive interface that simplifies the process of data gathering, allowing them to register to the application and insert their data in order to generate new menus.
- *Nutritionist Engagement and Menu Oversight:* Another important objective is the integration of nutritionists into the system's operation. Beyond incorporating nutritional control into the application, this also involves creating a platform for nutritionists to monitor one's client's saved menus but also giving them the option to create new specialized plans.

For a better definition and description of the main and secondary objectives of this license thesis we can enumerate some of the main requirements that are crucial for this application. Furthermore, we can dissect these requirements into functional and non-functional requirements which ensure that the system not only meets quality standards but also complies with user expectations.

The following functional requirements can be smoothly incorporated into the project objectives as follows:

- *Menus management by the user/nutritionists:* The application needs to offer a user interface for both the clients and nutritionist to easily manage their menus. Both should be able to save menus after the generation of new ones and also delete some of the already saved menus.
- *Shopping list management:* The application should offer the ability to save or delete certain food descriptions, by the clients, to their shopping list. This shopping list remains available for the clients in order to help them purchase products that they need for following the meals in real life.
- *Login/Register:* The application should provide this functionality to ensure security and privacy to our users no matter their type, nutritionists, or clients.
- *User creation and management:* The administrator should have the ability to perform certain operations on users, like update, delete or create new clients, or nutritionists, also the ability to assign clients to certain nutritionists.

The non-functional requirements have also been taken into account by these objectives, specifying in a clearer fashion the way the system works, rather than what it actually does like the functional requirements. Here are some worth mentioning non-functional requirements:

- *Usability:* To ensure that users can easily input their preferences and nutritional information the system should have an intuitive and user-friendly interface that is simple to navigate.
- *Performance:* To deliver real-time response and ensure a seamless user experience, the system should provide menu recommendation quickly especially as the user base expands.
- *Scalability:* The program must be able to handle large volume of data and an increasing number of users without sacrificing system responsiveness
- *Security:* To prevent unauthorized access and to comply with data protection laws, user data, particularly personal and dietary information must be securely stored.
- *Reliability:* To guarantee users have dependable access to their accounts and recommendations, the system should run continuously without frequent interruptions or downtime.
- *Adaptability:* The system should be able to adjust over time to accommodate dietary needs and user preferences, increasing its utility and relevance.



## Chapter 3. Bibliographic Research

The purpose of this chapter is to present the bibliographic studies that helped us choose our design decisions for implementing this recommender system that not only generates healthy meal plans for users that want to change their diet choices, but also highlight the resources used in our research process. The research process can be further broken down into a few essential elements such as: the investigation of how a personalized nutrition along with diet needs of a particular person that can ensure a good health and how can these needs be achieved and set, the study of choosing the right food composition dataset for the system, the examination of how recommender systems are used in current technologies, the research of a good MCDA algorithm that best suits our general classification of foods regarding their composition and well use for each person and the examination of how machine learning algorithms can be used for nutrition.

### 3.1. Personalized nutrition

As an introduction to the personalized nutrition study, we must keep in mind that for achieving a healthy diet plan for an individual we must take into consideration the uniqueness of its nutritional needs. In the paper [1], it was discovered how different people have particular responses to certain dietary components, this being the foundation and motivation for creating personalized nutrition strategies. The article also highlights the impact of one's unique attributes to their dietary plan and what key factors we need to take into account for providing the best nutritional strategy for each person, key factors such as age, height, weight, diseases or health status and also weight goals. I used all these key factors in order to compute, for each user, their nutritional profile and their Basal Metabolic Rate (BMR) in order to dynamically set the maximum calorie intake for each individual, but also the Body Mass Index (BMI) in order to classify them.

#### 3.1.1. Body Mass Index (BMI) and Basal Metabolic Rate (BMR)

For a clearer perspective on how the BMI affects and better describes the nutritional status of a particular person, I found that the article [2] show exactly how different BMI values can better classify each individual, than just relying only on the weight values. This paper also presents us the categories of the body mass index, showing us, in the *Figure 3.1 Categories of BMI*, the values that define each category.

TABLE 2 Categories of BMI	
Underweight	15–19.9
Normal weight	20–24.9
Overweight	25–29.9
Preobesity	
Class I obesity	30–34.9
Class II obesity	35–39.9
Class III obesity	≥40
Abbreviation: BMI, body mass index	

*Figure 3.1 Categories of BMI* [2]

After carefully defining the BMI value, we can use that alongside with the BMR to carefully specify the calorie intake that a person should consider. The basal metabolic rate, a person has, its computed with the Harris-Benedict formula shown in the article [3]. This formula also takes into consideration the activity level an individual has, this activity level, also presented in the mentioned paper, can be described as follows:

- *Sedentary*: minimal or no exercise, multiply your BMR by 1.2
- *Lightly active*: exercise lightly (once or twice a week), multiply your BMR by 1.375
- *Moderately active*: exercise moderately (3 or 5 times a week), multiply your BMR by 1.55
- *Very active*: exercise almost every day of the week, multiply your BMR by 1.725
- *Extra active*: exercise twice a day, every day, multiply your BMR by 1.9

The remaining value now represents the calorie intake that a particular person has to consume on a daily basis.

### 3.1.2. Nutritional strategies and dataset

We must take into consideration that every individual has a particular nutritional strategy to follow. As mentioned above, the height, weight and age are critical factors in computing one's calorie intake, as well as activity level and many more. The system is required to generate recommendation that are appropriate for users regarding not only the calorie intake, but also the quantity of important nutrients, key factors that affect our health and overall well-being. The paper [4] presents the importance of food compositions and their different nutrients, such as proteins, fats, carbohydrates, sugars, cholesterol and levels of sodium in 100 grams. Reading further the article I came across a sub-ontology (*Figure 3.2 The Nutritional Assesment sub-Ontology*) that gave me a better understanding on how combined foods should be prepared and presented in the final menu. Along side this knowledge, the paper [5] recommends to take into consideration different nutritional conditions for each individual such as diabetes or other abnormalities and not only the healthy cases. These crucial insights motivate us to explore a dataset that not only provides a wide range of edible products but also includes their nutritional compositions. This information is later utilized to determine whether a specific entry is suitable for an individual's needs. For this purpose, I have identified that the USDA National Nutrient Database for Standard Reference [6] dataset aligns well with our requirements.

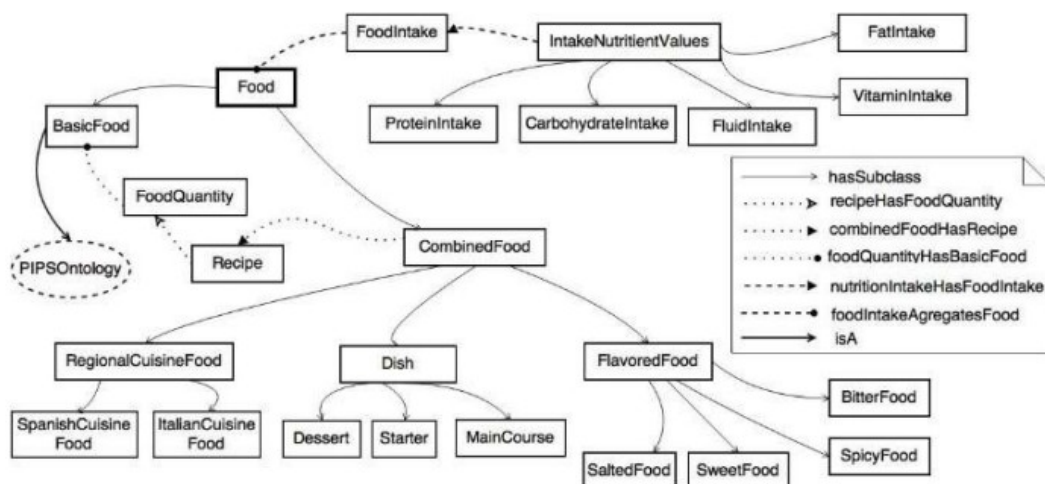


Figure 3.2 The Nutritional Assesment sub-Ontology [4]

## 3.2. Recommender systems

The paper [7], gave me a good understanding regarding the definition and the motivation behind the use of recommender system in our era. The article illustrates that the main need behind such systems is our lack of sufficient knowledge for making decisions autonomously and how we struggle to evaluate multiple suggestions due to our limited understanding of certain domains. In the context of nutrition and meeting dietary needs of an individual, I concluded from the paper [7] that such a system would be an essential component for giving our users suitable recommendations, due to our ignorance and lack of comprehension regarding of how crucial proper nutrition is to our well-being.

Another interesting fact was found in the papers [8] and [9] that shows how recommender systems reached new levels of popularity, over the last decades, in multiple domains, such as e-commerce, e-learning and e-tourism. The [8] article, also highlights that the most important and common task that the recommender system are designed to perform are the prediction of an item that belongs to a large array of objects with the use of one's individual preferences and the recommendation of the most suitable items for the user.

## 3.3. Multi-criteria decision analysis (MCDA)

As mentioned in the previous section, one of the most important requirements for implementing a good recommender system is to be able to make decisions based on a variety of criteria. Each of these systems must do a multi-criteria decision analysis using the appropriate tools to make the most suitable decision for any individual by taking into consideration all the needed criteria. So, are the multi-criteria decision-making (MCDM) tools useful? The answer to this question is highlighted and broken down in the paper [10], where the authors noted that while the MCDM techniques used for the same task have some variances, they are nonetheless useful and dependable. The article also shows and examines the algorithms behind some of the most reliable MCDM tools, such as Simple Multi-Attribute Rating Technique (SMART), Measuring Attractiveness by a Categorical Based Evaluation Technique (MACBETH) and Analytic Hierarchy Process (AHP).

The paper [9] also highlights the advantages of having an MCDM technique for a food recommendation system, by presenting step by step the integration of the AHP in the recommender algorithm. The goal of the AHP algorithm is to speed up the decision-making process by systematically evaluating and prioritizing multiple criteria. It also involves creating a hierarchical framework to divide a problem into manageable parts, determining relative importance through pairwise comparisons, and arriving at a final ranking. A simplified version of the algorithm is described below with the following steps, that are also highlighted by Gwo-Hshiung Tzeng and Jih-Jeng Huang in the book [11]:

1. **Hierarchical Setup:** Begin by breaking down the complex problem into a hierarchy of connected parts. offers a better visualization of this step.
2. **Comparative Weight Comparison:** This step involves the comparison of the relative weights, or the priority of the attributes associated with the decision-making aspects, by the use of a reciprocal comparison matrix.
3. **Subject Judgement Synthesis:** Calculate the relative weights of the attributes using individual judgements. A more comprehensive understanding of the significance of each attribute can be achieved by taking into account a variety of inputs.
4. **Weight Aggregation:** This step requires the aggregation of the estimated relative weights of elements to determine the best alternatives or strategies.

Based on the specified criteria and hierarchy, you can order and choose the best options using this weights synthesis.

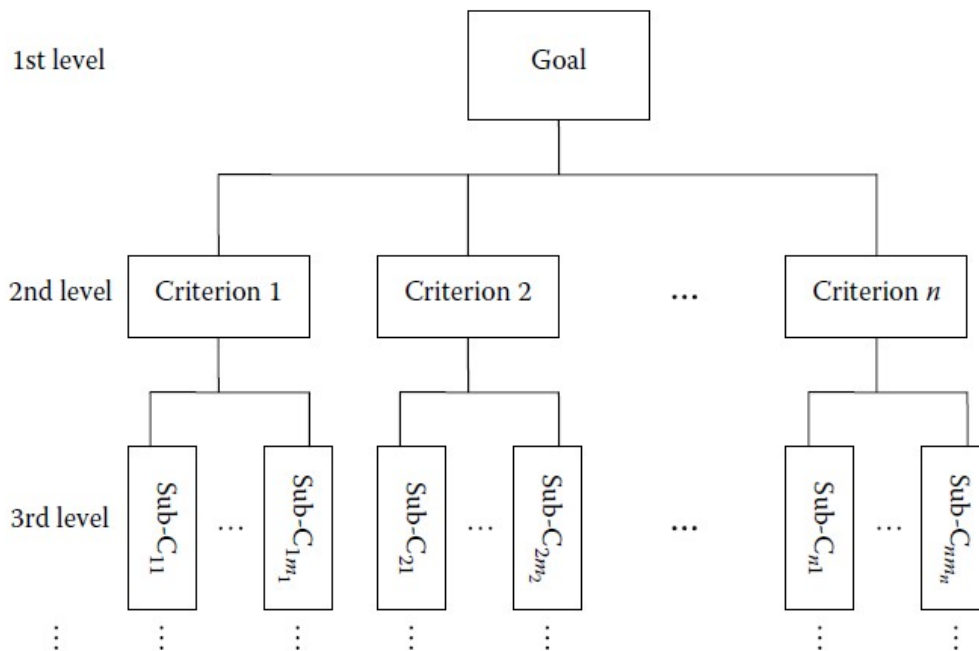


Figure 3.3 The hierarchical structure of AHP [11]

### 3.4. Machine Learning (ML)

In the field of machine learning, the choice of a suitable algorithm for a customized food recommender system is crucial, given that the system wants to smoothly incorporate user preferences into its recommendations. Machine learning algorithms provide an effective and powerful technique to achieve this objective and provide a healthy meal plan for our user.

This study offered us a better understanding of the machine learning technologies and how they can be vital in the development of such systems, and they can be broken down into multiple key elements: the research of the meaning behind the terminology of “Machine Learning” and the types of machine learning technologies, the study for the applications of such technologies and algorithms in healthcare and nutrition.

#### 3.4.1. Introduction

As the article [12] suggest, machine learning is a key component of artificial intelligence (AI) and enables systems to learn from experience and evolve without explicit programming. ML can be further broken down into 3 main categories that are recognised within this field: Supervised Learning, Unsupervised Learning, and Reinforcement Learning.

When training a model through supervised learning, it is important to distinguish between classification, where the output is a category, and regression, where the output is a true value . Contrarily, unsupervised learning uses unlabeled data to find complex patterns and relationships and is frequently used in tasks like dimension reduction and clustering

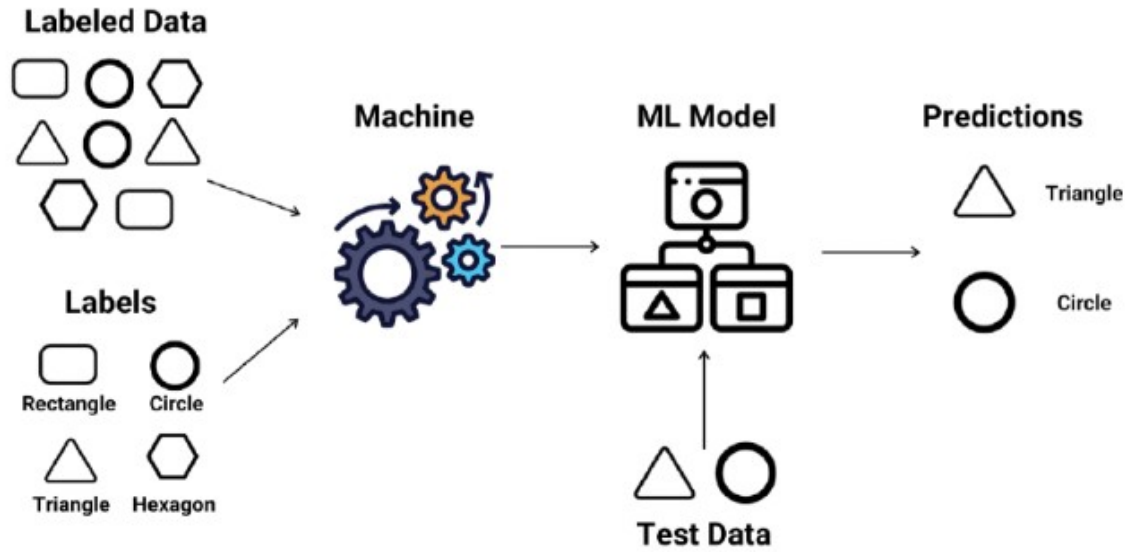


Figure 3.4 Illustration of the supervised learning [12]

To improve predictions, reinforcement learning uses agents, environments, states, actions, and incentives. A decision-making process, an error-correcting function, and a model optimization process are all components of learning. Several tools are used in the field of machine learning algorithms to support several applications. For pattern recognition, neural networks replicate the human brain, whereas linear regression predicts numerical values using linear relationships. Clustering identifies data patterns, decision trees offer a visual depiction of data categorization and regression, and logistic regression deals with categorical responses. For prediction, random forests combine the results of various decision trees. These machine learning technologies and algorithms are vital in contemporary technological landscapes due to their adaptability and importance.

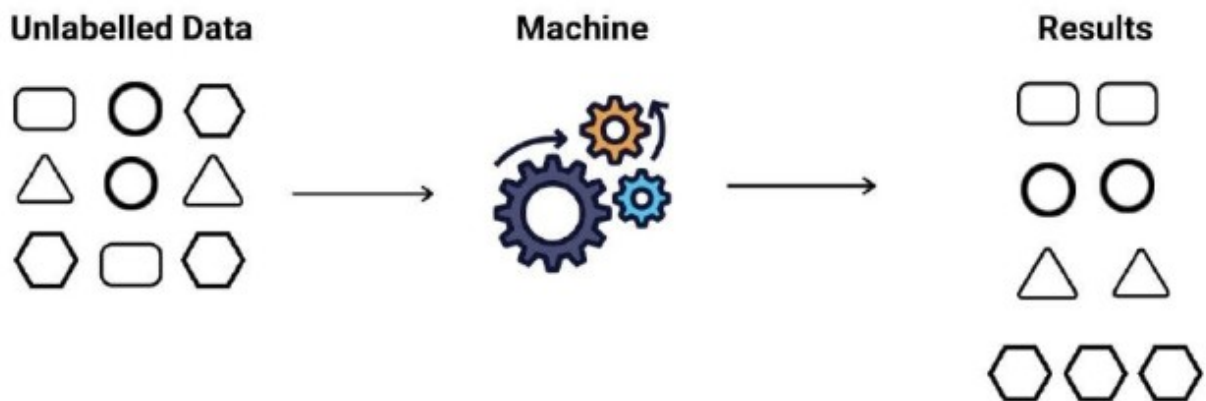


Figure 3.5 Illustration of the unsupervised learning [12]

#### 3.4.2. Machine Learning in healthcare and nutrition

The paper [13] show that in the realm of healthcare, the large amount of information has traditionally been kept in tangible records kept at medical facilities. The switch to electronic medical records (EMR) is, however, revolutionising healthcare data, making it possible for AI technologies to make use of this wealth of knowledge. The diversity of healthcare information further exacerbates the growth of electronic data, often known as Big Data. Integral parts include medical pictures, patient histories, clinical outcomes, and genetic databases like the UK Biobank and the Cancer Genome Atlas (TCGA). When it comes to the structure, use, and processing of data, these databases provide special issues. The emergence of machine learning has a significant impact on the healthcare industry, as AI applications are addressing a variety of issues such medical image processing. Despite the promise applications, incorporating machine learning into clinical settings is done so cautiously due to worries about patient safety, legal issues, and the requirement for strong validation mechanisms.

Genetic research has changed as a result of the Next-Generation Sequencing (NGS) revolution and the exponential expansion of genetic data. The sequencing of the entire human genome has been sped up because to these technologies, which have also greatly decreased prices. Understanding gene interactions is complicated by the complexity of the human genetics and the differences between people. A huge section of the genome, known as genetic dark matter or missing heritability, lacks protein-coding function and is the subject of several large-scale genetic research that seek to find trends within populations. This genetic "dark matter" could affect how genes are expressed, which could have negative effects on health. In order to understand these intricacies, machine learning techniques have become essential for spotting patterns and trends in the massive genetic data. Machine learning helps to uncover genetic correlations and provides insights into complex disorders like schizophrenia and bipolar disorder. It also has the ability to forecast disease risks, such as those for cancer and Alzheimer's disease.

Another interesting application of the machine learning is carefully examined and discussed in the paper [14]. As the article focusses mostly in metabolomics and personalized nutrition, shows that there is a popularity growth in this field, due to its capacity to profile a wide variety of metabolites within samples. Preprocessing and analysis are required for this enormous dataset, tasks that machine learning is well suited for. Metabolomics is used to examine food objectively by finding dietary biomarkers and their relationships to health outcomes. Disease prediction, phenotyping, biomarker identification, and evaluation of food patterns are all made possible by machine learning techniques including clustering and classification. Examples include identifying good and unhealthy phenotypes, identifying dietary variations, and contrasting the metabolic activities of the microbiota. In order to navigate the complexity of metabolomics data and gain insights into the effects of nutrition on health, machine learning's capacity for pattern identification and feature selection comes in very handy.

### 3.5. Related works

This chapter provides a thorough examination of the academic works that have aided in the improvement of nutritional sciences and the development of personalized nutritional recommendations.



### 3.5.1. DIETOS: a recommender system for health profiling and diet management

One worth mentioning related works is presented in the article [19]. As the paper presents, DIETOS (DIET Organiser System) is a web-based Recommender System (RS) that caters to a wide range of users, including those who live healthy lives as well as those who have chronic health issues such as CKD, diabetes, and hypertension. This comprehensive system uses user health profiles to give personalised nutritional recommendations that take into consideration the region's specific dietary unique characteristics.

DIETOS involves users in a series of medical enquiries that cover many elements of their health in order to develop a user's health profile. These questions include a wide variety of factors, such as test findings and vital signs. DIETOS produces an adaptive health profile that changes with the user's input by collecting and analysing answers over time. This adaptive method guarantees that consumers receive recommendations that are relevant to their changing health situations. For a better understanding on how this system is structured, we can examine the architecture illustrated in *Figure 3.6 DIETOS Architecture* [19].

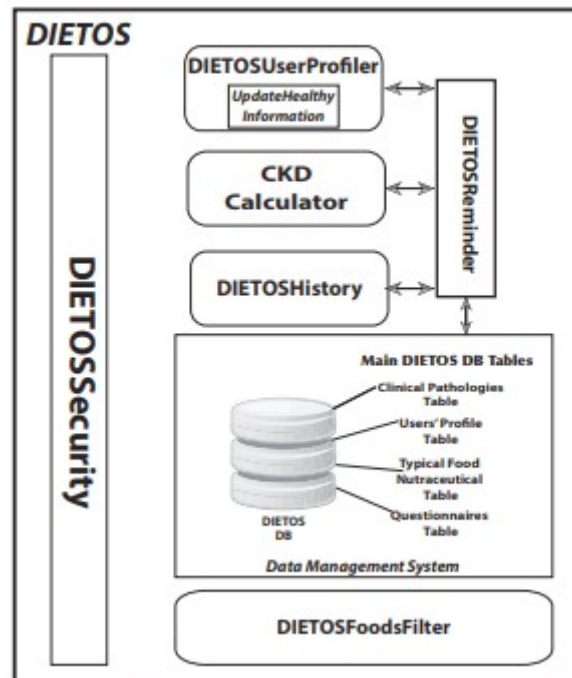


Figure 3.6 DIETOS Architecture [19].

### 3.5.2. A cross-domain framework for designing healthcare mobile applications

The main problem that this paper tries to address is the inadequate monitoring and guidance for athletes or other individuals that are engaged in physical activities. The article [20] shows that even with wearable sensors and mobile applications available there still exists a lack of comprehensive and personalized recommendations among these tools. That said, the article offers an approach that includes the use of a social semantic mobile framework that blends user-generated data with government health information and expert expertise. It is concerned with assessing physical conditions, developing customised food and workout programmes, monitoring and recalculating plans, and adjusting the amount of demand in nutrition and exercise plans.

The methodology proposed by this article involves 3 major steps:

- Healthcare Data Gathering Phase: which involves a nutritional assesment of the user, along with the physical evaluation and user profile identification.
- Data Semantic Analysis and Generation of Healthcare Plans Phase: where the system analyzes the collected data and geneerates the required training and nutritional plans.
- Adjustment of Plans and Combination of Social Networks Phase: where the monitoring of the plans and user evaluations take place.

A team of professionals and users evaluated the created training and diet regimens during the validation phase. The efficacy rates for training session plans were 82% and 86%, respectively, with an overall effectiveness rate of 81%. Validation was carried out using health and sports measurements derived from government standards and recommendations.

### 3.5.3. SousChef: Mobile Meal Recommender System for Older Adults

The article [21] introduces the SousChef meal recommender system in order to combat the complex challenge, that is particularly found among older people, of adopting a healthy diet in today's large array of diverse food options.

SousChef's food recommendations and application interface have been meticulously created to correspond with the tastes and needs of older persons, including user-friendly interfaces and adhering to nutritionist advise. Comprehensive testing involving actual users were undertaken to confirm the system's efficacy and user-friendliness.

The SousChef system (*Figure 3.7 Overview of SousChef Recommender System [21].*) is made up of a central cloud server and a mobile application that serves as the system's user interface. The cloud server is in charge of centrally storing system information and making it accessible via web service application programming interfaces (APIs). Its ease of use also makes it easier to integrate information from other sources, as exemplified by the integration with Fitbit cloud services to obtain users' activity metrics recorded by Fitbit [22] devices.

Given the computational requirements for creating meal recommendations, the server's better processing capabilities as compared to mobile devices also makes it a more acceptable alternative. Mobile devices can use web service APIs to initiate the development of suggestions.

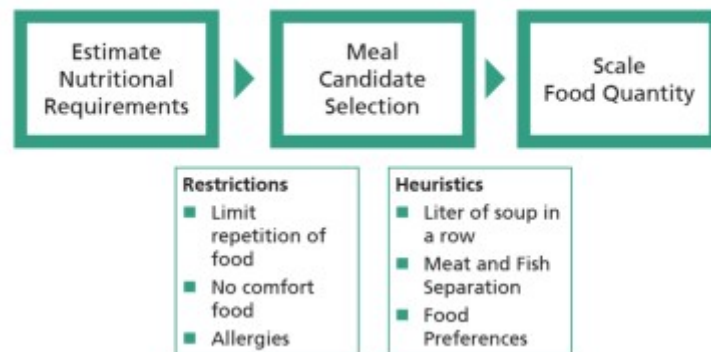


Figure 3.7 Overview of SousChef Recommender System [21].



## Chapter 4. Analysis and Theoretical Foundation

An extensive investigation and a sound theoretical base are essential for developing a reliable and successful food recommendation system. This chapter explores the intricate workings of the multiple components that make up the core of our food recommendation application. Starting with a thorough explanation of the dataset that serves as the foundation for the intelligence of our application, we will now examine the important factors that influence the recommendations made by our system, such as presenting the important steps and examine the core of the Analytic Hierarchy Process, along with the K-Nearest Neighbors machine learning algorithm and the evaluation metrics used for such a recommender system.

### 4.1. The Description of the Dataset

As mentioned before, the key component for developing an efficient, but also reliable meal recommender system, is choosing the most suitable dataset. The sheer variety and complexity of each person's nutritional needs presents a significant problem for our design. Our app need access to a comprehensive dataset that captures the unique characteristics of various food products, their nutritional composition, and their suitability for various dietary profiles in order to achieve a compatibility with different dietary profiles.

As previously stated, I found that the most suitable dataset for achieving our goal and set a good starting point for this design is the USDA (United States Department of Agriculture) National Nutrient Database for Standard Reference [6]. The primary source of information on food composition in the United States is the USDA, which also serves as the basis for the majority of food composition databases in both the public and private sectors. It compiles knowledge regarding various food categories worth of macronutrients, micronutrients, vitamins, minerals, and other nutritional components. Foods are divided into several groupings according to the USDA SR's elaborate categorization system, which uses botanical, common, or scientific names. This taxonomy makes it easier to precisely explore food products and find alternatives that suit customers' preferences and dietary needs.

ID	Description	Calories	Protein	TotalFat	Carbohydride	Sodium	Saturated	Cholesterol	Sugar	Calcium	Iron	Potassium	VitaminC	VitaminE	VitaminD
1001	BUTTER, WITH SALT	717	0.85	81.11	0.06	714	51.368	215	0.06	24	0.02	24	0	2.32	1.5
1002	BUTTER, WHIPPED, V	717	0.85	81.11	0.06	827	50.489	219	0.06	24	0.16	26	0	2.32	1.5
1003	BUTTER OIL, ANHYDR	876	0.28	99.48	0	2	61.924	256	0	4	0	5	0	2.8	1.8
1004	CHEESE, BLUE	353	21.4	28.74	2.34	1395	18.669	75	0.5	528	0.31	256	0	0.25	0.5
1005	CHEESE, BRICK	371	23.24	29.68	2.79	560	18.764	94	0.51	674	0.43	136	0	0.26	0.5
1006	CHEESE, BRIE	334	20.75	27.68	0.45	629	17.41	100	0.45	184	0.5	152	0	0.24	0.5
1007	CHEESE, CAMEMBER	300	19.8	24.26	0.46	842	15.259	72	0.46	388	0.33	187	0	0.21	0.4
1008	CHEESE, CARAWAY	376	25.18	29.2	3.06	690	18.584	93		673	0.64	93	0		
1009	CHEESE, CHEDDAR	403	24.9	33.14	1.28	621	21.092	105	0.52	721	0.68	98	0	0.29	0.6
1010	CHEESE, CHESHIRE	387	23.37	30.6	4.78	700	19.475	103		643	0.21	95	0		
1011	CHEESE, COLBY	394	23.76	32.11	2.57	604	20.218	95	0.52	685	0.76	127	0	0.28	0.6
1012	CHEESE, COTTAGE, C	98	11.12	4.3	3.38	364	1.718	17	2.67	83	0.07	104	0	0.08	0.1
1013	CHEESE, COTTAGE, C	97	10.69	3.85	4.61	344	2.311	13	2.38	53	0.16	90	1.4	0.04	0
1014	CHEESE, COTTAGE, N	72	10.34	0.29	6.66	330	0.169	7	1.85	86	0.15	137	0	0.01	0
1015	CHEESE, COTTAGE, L	86	11.83	2.45	3.66	330	0.979	10	3.67	91	0.15	84	0	0.04	0
1016	CHEESE, COTTAGE, L	72	12.39	1.02	2.72	406	0.645	4	2.72	61	0.14	86	0	0.01	0
1017	CHEESE, CREAM	342	5.93	34.24	4.07	321	19.292	110	3.21	98	0.38	138	0	0.29	0.6
1018	CHEESE, EDAM	357	24.99	27.8	1.43	965	17.572	89	1.43	731	0.44	188	0	0.24	0.5
1019	CHEESE, FETA	264	14.21	21.28	4.09	1116	14.946	89	4.09	493	0.65	62	0	0.18	0.4
1020	CHEESE, FONTINA	389	25.6	31.14	1.55	800	19.196	116	1.55	550	0.23	64	0	0.27	0.6
1021	CHEESE, GIETOST	466	9.65	29.51	42.65	600	19.16	94		400	0.52	1409	0		
1022	CHEESE, GOUDA	356	24.94	27.44	2.22	819	17.614	114	2.22	700	0.24	121	0	0.24	0.5

Figure 4.1 A small section of the USDA National Nutrient Database for Standard Reference

As illustrates, the dataset presents each food entry along with their nutritional composition: calories, macronutrients, micronutrients, and vitamins. As [6] suggests, the dataset contains data on 7,793 food items and 15 attributes that will further be used in the AHP algorithm as criteria. The attributes that describe each food item are the following:

- *Description*: Represents the name of the food products, each having a unique string value.
- *Calories*: Represents the amount of energy that each food product provides in 100 grams, measured in calories.
- *Protein*: Represents the amount of proteins that are contained in 100 grams of each food product, measured in grams.
- *Total Fat*: Represents the value of the fat intake in 100 grams of food, measured in grams.
- *Carbohydrates*: Represents the amount of carbohydrates found in 100 grams of each food item, also measured in grams
- *Sodium*: Represents the sodium level and intake of each food item found in 100 grams of food, measured in milligrams.
- *Saturated Fats*: Represents the value of saturated fats found in 100 grams of each food product, this value is also contained in the value of Total Fat and measured in grams.
- *Cholesterol*: Represents the amount of cholesterol found in each food item, measured in milligrams.
- *Sugar*: Represents the level of sugars that each food item provides, measured in grams.
- *Calcium*: Represents the amount of calcium found in the composition of each food product, measured in milligrams.
- *Iron*: Represents the level of iron each food product has in its composition, measured in milligrams.
- *Potassium*: Represents the amount of potassium that is contained in 100 grams of each food item, measured in milligrams.
- *Vitamin C, E, and B*: each vitamin has its own column and represents the number of vitamins found in each's 100 grams of food entry, measured in milligrams.

After a careful analysis of the nutritional attributes presented above, I concluded that all of them are important to our nutrition, but to reduce the system complexity, I took into consideration only the following: Calories, Proteins, Total Fat, Carbohydrates, Sodium, Saturated Fat, Cholesterol and Sugar. Furthermore, we use each of these attributes to determine their priority by comparing them with each other and also with every user's dietary needs.

## 4.2. Analytic Hierarchy Process (AHP)

Another core element that helps us fortify the foundation of our application is the Analytic Hierarchy Process, a MCDM analytical framework that helps us classify the appropriate food items, from the inappropriate, regarding each's individual nutritional needs.

As an addition to the information presented in the bibliographic study segment, the book [15] highlights the cornerstone of the AHP and carefully examines the process behind it. The book presents that the AHP was created by T. L. Saaty at the Wharton School between 1971 and 1975 as a framework for making judgements based on paired element comparisons. It focuses on generating ratio scales from paired comparisons that are both discrete and continuous to reflect the relative strength of preferences and emotions. The importance of

AHP resides in its capacity to support both inductive and deductive reasoning at the same time, considering many elements and allowing for interdependence and feedback.

#### 4.2.1. The Principles and Axioms of AHP

Another important aspect that the book [15] illustrates are the 3 main principles for solving a problem using the Analytic Hierarchy Process: decomposition, comparative judgements and synthesis of priorities.

1. The decomposition principle is a key step in organising complex problems in a hierarchical fashion. With this method, difficult problems are divided into manageable chunks, starting with broad objectives and progressively moving on to more detailed criteria and subcriteria. It is emphasised that the distinction between functional and structural dependence, made by T. L. Saaty, allows for a wide range of measuring options.
2. The comparative judgement principle represents another vital step and describes in detail the importance of pairwise comparisons. These comparisons are used to determine the relative relevance of pieces within a specific level, with an emphasis on shared criteria or features from the level above. The use of matrices and primary eigenvectors allows for the quantification of element priority.
3. The synthesis of priorities principle requires the synthesis of priorities throughout the hierarchical structure, which involves multiplying local priorities by the priority of their equivalent criteria in the level above. This cumulative method takes into consideration element dependencies and interconnections, resulting in composite or global priorities. The subsequent cascading effect aids in the refinement of priorities at many stages.

The AHP is based on four axioms that serve as the guiding steps for its implementation. These axioms serve as the foundation for AHP's logical and mathematical framework, ensuring consistency and reliability in decision-making. AHP's four axioms are as follows:

- **Reciprocal Axiom:** Highlights the essential principle of reciprocal comparisons. It claims that if one aspect is judged more significant than another, then the converse must also be true. If element A is preferred over element B, then the reciprocal preference of B over A should also be valid. This axiom ensures logical coherence in decision-makers' pairwise comparisons.
- **Homogeneity Axiom:** Is based on the concept of consistency between comparisons. It claims that if one element is thought to be more essential than another, and that other element is thought to be more important than a third, then the first element must likewise be thought to be more important than the third. In other words, if A is preferred over B and B is chosen over C, A should be preferred over C. This axiom ensures that preferences across elements stay consistent and logical.
- **The Axiom of Expectations:** Describes the concept that the measurement scale used for comparisons should have no bearing on the end result. AHP enables subjective judgements on many scales, such as numerical or verbal. This axiom ensures that, regardless of the scale used, the relative priority resulting from pairwise comparisons remain constant. It demonstrates AHP's adaptability in accepting a wide range of tastes and measuring scales.
- **Independence Axiom:** Is concerned with the choice problem's hierarchical nature. It states that the existence or absence of other factors at the same or any other level

should not impact the relative value of two elements at a lower level. In other words, priorities within a level should be determined independently of the context or items in higher levels. In AHP, this axiom promotes the concept of structured and hierarchical decision-making.

#### 4.2.2. The Phases of AHP

Here we will further discuss the main algorithm behind the Analytic Hierarchy Process (AHP). For a better classification of the appropriate foods from all of the entries of the dataset, we used the AHP for a careful examination of each food product according to their nutritional composition and current user's dietary needs. A better understanding of the algorithm, offered by the paper [9], is shown in Figure 4.2 The AHP Model and further broken down below.

The algorithm resumes on having a total of 8 main steps that are further classified in 3 major phases:

- Phase 1: The definition of the problem and goal:
  - Step 1: We establish our criteria  $c_j$ , where  $j = 1, 2, \dots, m$ , the alternatives  $a_k$ , where  $k = 1, \dots, l$  and we also establish the goal.
  - Step 2: We establish the final classes  $C_i$ , where  $i = 1, \dots, n$  in which we want to classify the alternatives, for our case it would be only 2 classes: appropriate and inappropriate.
  - Step 3: We establish the local limiting profiles  $lp_{ij}$ , which define each class  $C_i$ , which represent the minimum performance that a criteria should have to belong to a class.
- Phase 2: The evaluation of the entries:
  - Step 4: We assign a priority to each criterion  $c_j$ , and compute their weights  $w_j$ , using the AHP eigenvalue approach. The eigenvalue method revolves around the solving one of the following equation:
 
$$A \cdot p = \lambda \cdot p, A \cdot w = \lambda \cdot w,$$

$A$  – comparison matrix;  
 $p$  – priority vector;  
 $w$  – weight vector;  
 $\lambda$  – maximal eigenvalue.
  - Step 5: We compare in a pairwise manner each alternative  $a_k$ , with the limiting profile  $lp_{ij}$ , of each criterion  $c_j$ .
  - Step 6: From the matrices that resulted from the last steps, we compute the local priority  $p_{kj}$ , for each alternative and the local priority for each limiting profile  $p_{ij}$ , with the eigenvalue method.
- Phase 3: The classification:
  - Step 7: We compute the global priorities, of every alternative  $a_k$  ( $p_k$ ) and of every limiting profile  $lp_{ij}$  ( $lp_i$ ), by summing up the products of each's local priority with their corresponding criteria weight  $w_j$  as shown below in the (1) and (2).

$$p_k = \sum_{j=1}^m p_{kj} w_j \quad (1)$$

$$lp_i = \sum_{j=1}^m p_{ij} w_j \quad (2)$$

- Step 8: We compare the values computed at the step 7 and classify each alternative to their resulted class. We repeat the steps 5 to 8 for every alternative in the dataset.

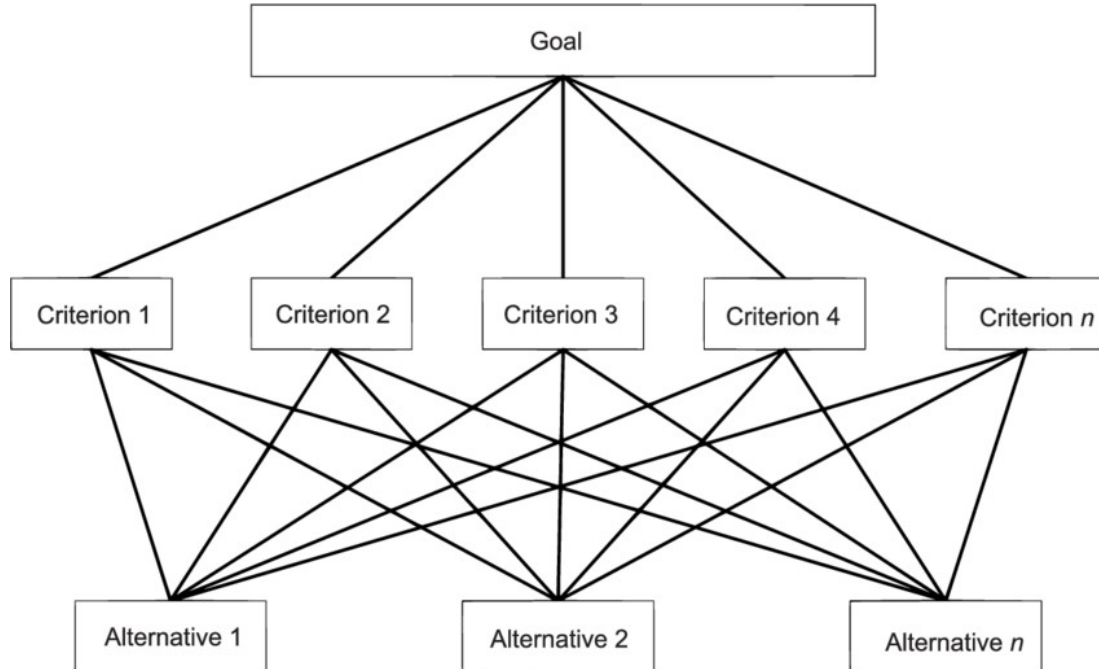


Figure 4.2 The AHP Model

### 4.3. K-Nearest Neighbors (KNN)

We start the analysis of the K-Nearest Neighbors as a continuation of the bibliographic study of machine learning. KNN is a basic yet powerful instance-based learning algorithm that falls within the supervised learning category. It is widely utilized in different sectors, including healthcare and nutrition, for classification and regression problems. The implications of the KNN with our specific goal, implementing a reliable food recommender system, revolve around the integration of preferences to our meal planning by taking into account the nutritional composition of each food item.

#### 4.3.1. Introduction to KNN

The paper [16] offers us a good understanding of the K-Nearest Neighbors classifier and presents us the very core and foundation of the algorithm along with the mathematical considerations. The paper shows that the Nearest Neighbour Classification is actually straightforward, it classifies each example based on the closest class. This strategy can be extended to take into account numerous neighbours, giving rise to the k-Nearest Neighbour (k-NN) Classification technique, which determines the class based on the k nearest neighbours.

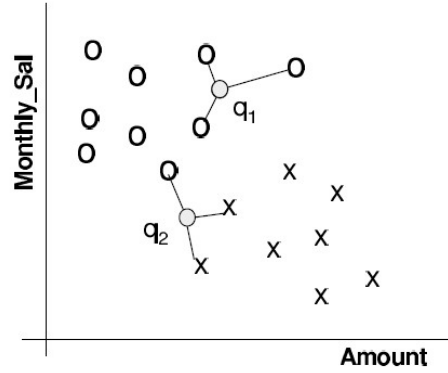


Figure 4.3 A 3-NN Classification [16]

Because training samples must be kept in memory during runtime, this method is also known as Memory-Based Classification. It falls under the heading of Lazy Learning because it makes decisions at runtime and also because it classifies directly based on training examples, it is also known as Example-Based Classification or Case-Based Classification. The authors of this paper offer us an easy to understand example (see ) that illustrates how a 3-Nearest Neighbors algorithm works for 2 classes (O and X) and a 2D feature space. The result of this example shows that the class assigned to  $q_1$  is O, but the case of  $q_2$  is a little more complex and can involve different voting methods in order to determine which class should it be assigned to.

#### 4.3.2. Mathematical Background of the KNN

In this section, we will go deeper into the main process underlying our KNN algorithm. With the help of the article [17], we formed a better understanding of the mathematical characteristics of the KNN. The author points out that one of the main characteristics of this machine learning classifier is the Euclidean distance.

The Euclidean distance is an essential concept in geometry that is widely used to determine the similarity and dissimilarity of two points in a Euclidean space.. Here it is described as the distance between the training samples and a test sample, defined by the next formula:

$$d(x_i, x_l) = \sqrt{\sum_{j=1}^p (x_{ij} - x_{lj})^2}$$

To further break down the formula,  $x_i$  and  $x_l$  represent two input samples from the input sample space, where  $i, l = 1, 2, \dots, n$ , and  $n$  is the number of total input samples. The number of total features is represented by  $p$ .

Another important aspect that it is discussed in [17], is the classification decision rule. The paper suggests that the KNN algorithm's classification determination rule is based on finding the class label for a test sample using its nearest neighbors in the training data. The procedure entails comparing data point distances and classifying the test sample based on the majority class of its  $k$  nearest neighbours. The process can be further broken down by the following steps:

1. *Train/Test/Split*: The dataset is separated into two categories: training samples and test samples. The class labels of training samples are utilised to create the classifier during training. The class labels of test samples are predicted during testing based on their neighbours in the training data.
2. *1-Nearest Neighbor Rule*: The projected class of a test sample is set equal to the class of its nearest neighbour in the 1-nearest neighbour case ( $k=1$ ). If the nearest neighbour has the class label  $c_m$ , then the test sample is also categorised as  $c_m$ .



3. *k-Nearest Neighbors Rule*: For  $k$ -nearest neighbours ( $k > 1$ ), a test sample's predicted class is assigned to the class that appears most frequently among its  $k$  nearest neighbours. This rule considers the neighborhood's predominant class and allocates the test sample to that class.
4. *Confusion Matrix and Accuracy*: During testing, the confusion matrix ( $C$ ) is utilised to tabulate the predicted class labels of test samples. The matrix has a dimension of  $\Omega \times \Omega$ , where  $\Omega$  represents the number of total classes. If the projected class is right, the diagonal element corresponding to that class is increased by one during testing. If the predicted class is wrong, the off-diagonal element representing the true and predicted classes is increased by one.
5. *Classification Accuracy*: Following the classification of all test samples, the classification accuracy is calculated as the ratio of the number of correctly categorised samples to the total number of samples classified.

#### 4.3.3. Feature Transformation Techniques

In the context of KNN classification, feature transformation refers to the act of changing or developing new features from existing ones in order to improve the performance of the KNN algorithm. This is especially useful when the original features are not directly sufficient for good classification or when specific patterns in the data are not visible in the original feature space. Feature transformation might entail a number of strategies, each of which serves a distinct purpose:

- **Normalization and Standardization**: These are typical preprocessing strategies for scaling features to a consistent range. Standardisation alters characteristics to have a mean of 0 and a standard deviation of 1, whereas normalisation scales them to a similar range. These strategies aid in preventing features with higher numerical ranges from dominating the KNN algorithm's distance calculations.
- **Dimensionality Reduction**: Datasets may include a huge number of features, which might cause the "curse of dimensionality" and impair KNN performance. Principal Component Analysis (PCA) techniques can minimise the dimensionality of the feature space while retaining the majority of the critical information.
- **Feature Engineering**: This includes designing new features based on existing ones. In picture categorization, for example, you might engineer texture, colour, or form attributes. These new features can detect patterns that were not seen in the original features.
- **Kernel Functions**: In some circumstances, feature modification is performed implicitly through the use of kernel functions. These functions implicitly translate the original features to a higher-dimensional space in which the data may be more easily separated. The linear, polynomial, and radial basis function (RBF) kernels are examples of common kernel functions.
- **Distance Metric Learning**: This requires learning a customised distance metric that adjusts to the data's individual properties. Rather than utilising the Euclidean distance by default, distance metric learning can alter the distances based on the relevance and value of the attributes.
- **Nonlinear Transformations**: These transformations can be applied to features in data that is not linearly separable. These modifications strive to generate more complicated and flexible decision boundaries.

Macronutrients and micronutrients, such as carbohydrates, fats, and proteins are frequently quantified in different units such grammes, calories, or percentages. Because these units are not directly comparable, employing them in a KNN algorithm as is can result in

biased results. For example, if one macronutrient is measured in grammes and another in calories, the algorithm may give the macronutrient with the higher numerical value greater weight, even if it is not fundamentally more significant. As this seems to be related to our case, the problem is addressed by normalisation and standardisation, which convert the features to a common scale and make them immediately comparable.

#### 4.3.4. The flow of the KNN algorithm

The k-Nearest Neighbours (k-NN) algorithm is a basic yet successful classification strategy that predicts the class of a test item based on its nearest neighbours class labels in the training dataset. Here's the step-by-step process of the algorithm:

1. **Data Preparation:** We collect and preprocess the dataset, which should include the features (attributes) of the items to be classified as well as their corresponding class labels.
2. **Chose the appropriate value of k:** We determine the number of neighbours (k) to take into account during classification. This can be done through trial and error or by employing cross-validation techniques to determine the best k for your dataset.
3. **Feature transformation:** As mentioned before this step is optional, but ensures a better functionality and precision of the algorithm by using one of the methods presented above.
4. **Distance calculation:** Using a distance metric, such as the Euclidean distance, we compute the distance between the test item and all training items. Based on their feature values, the distance metric determines how "similar" or "close" two items are.
5. **Nearest Neighbor Selection:** We choose the k training items that are closest to the test item.
6. **Majority Voting:** We determine the most frequent class by examining the class labels of the k-nearest neighbours. This will be the test item's projected class.
7. **Classification Decision:** Based on the majority class from the k-nearest neighbours, lastly we will assign the anticipated class label to the test item.

## 4.4. Evaluation Metrics

In the world of machine learning activities, evaluation metrics are critical. Different metrics address the special needs of classification and regression tasks. Some measures, such as precision-recall, are useful for a wide range of jobs. The supervised learning landscape includes classification and regression, two main kinds of machine learning. By including a variety of evaluation indicators, we improve our capacity to fine-tune a model's predictive performance prior to its deployment on novel, previously unknown data. When the model encounters fresh, previously unseen data, relying exclusively on accuracy for evaluation may prove insufficient, perhaps leading to inferior predictions. With the help of the article [18], in the next part, I will dig into the world of Classification evaluation metrics, providing light on their significance in improving the generalizability of machine learning classification models.

As mentioned before, predicting class labels based on incoming data is what classification is all about. Consider the well-known case of email spam detection, in which emails are classified as "spam" or "not spam". Metrics like accuracy, confusion matrix, log-loss, and AUC-ROC can be used to assess classification performance. Precision-recall is proven to be particularly useful in classification tasks.



#### 4.4.1. Accuracy

Accuracy is a key evaluation parameter used for measuring a classification model's performance. It denotes the fraction of successfully predicted instances in relation to the total number of examples in the dataset, or in relation with the total number of predictions. The following is the accuracy formula:

$$Acc_{score} = \frac{\text{No. of Correct Predictions}}{\text{No. of Total Predictions}}$$

A good example for illustrating how the accuracy can be interpreted is the "spam"/"not spam" classification of the emails. Suppose we have a test dataset of 200 emails and after performing a prediction on each one of them we conclude that the model correctly predicts 140 of them and misclassifies 60 emails. This gives us an accuracy score equal to 0.7, where in general, this values always lie in between 0 and 1, 0 representing no correct predictions and 1 represents that the model predicted correctly in all cases.

#### 4.4.2. Confusion Matrix

The confusion matrix is an important tool for evaluating the performance of machine learning classification algorithms. It's a structured table that shows how closely the model's predictions match the actual ground truth labels for various classes in a classification task. This matrix depicts the model's predictions as well as their relationship with the actual outcomes. The article [\[18\]](#) presents a good example () of the following 4 main categories of the confusion matrix:

- *True Positives (TP)*: These are the instances in which the model properly predicted the positive class and the actual label is positive. In simpler words, it denotes the model's success in detecting positive cases.
- *True Negatives (TN)*: These are situations in which the model correctly predicts the negative class but the actual label is negative. It represents the model's ability to detect negative events.
- *False Positives (FP)*: In these cases the model predicts a positive class while the actual label is negativ. This is known as a "Type 1 Error" and indicates the model's proclivity to generate false alarms.
- *False Negatives (FN)*: These occur when the model predicts a negative class while the actual label is positive. This is known as a "Type 2 Error" and shows that the model failed to recognise affirmative cases.

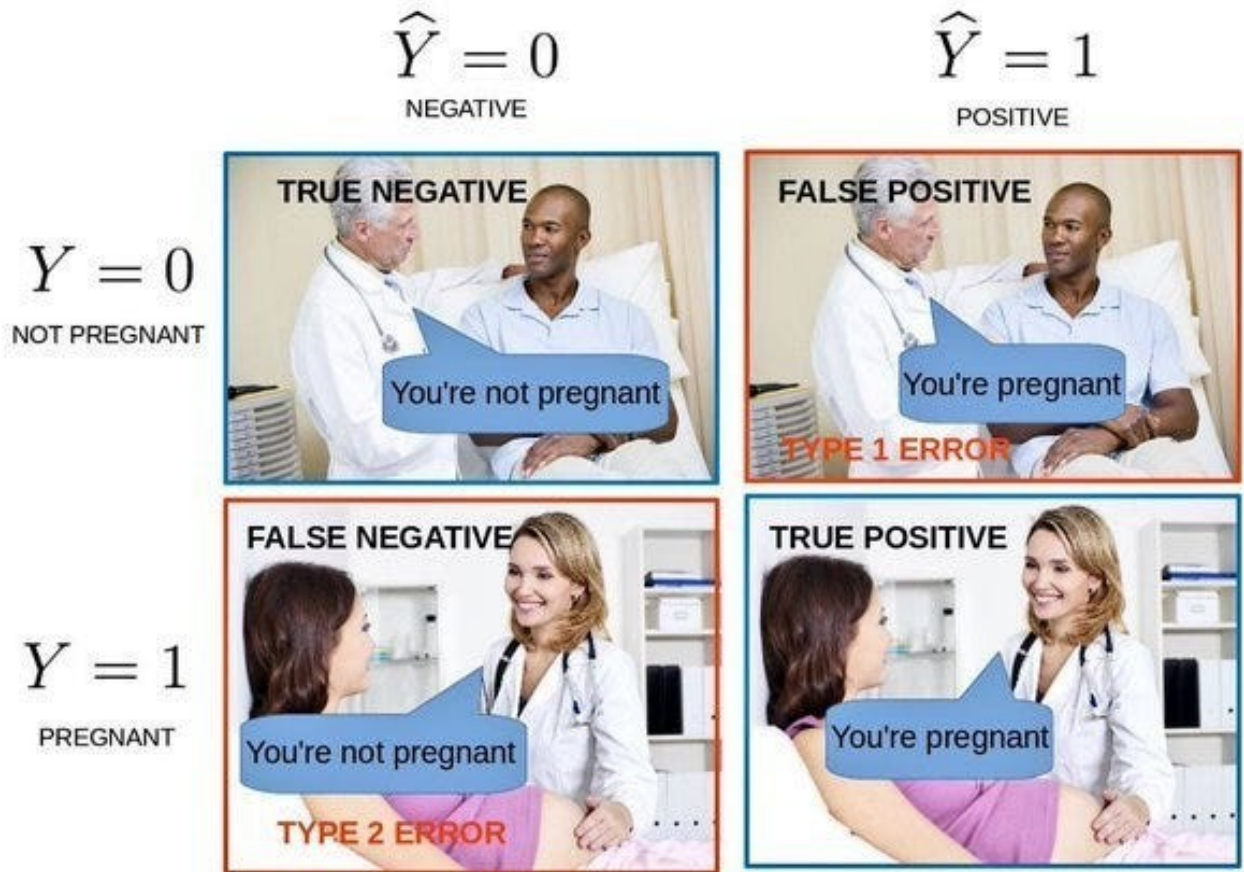


Figure 4.4 Confusion Matrix Example [18]

With these categories we can further explore the benefits of the confusion matrix along with the computation of other evaluation metrics such as precision, recall and f1 score:

- **Precision:** Precision is the fraction of accurately predicted positive cases out of all positive cases predicted by the model. Precision is especially important when False Positives are a worry, such as when making inaccurate optimistic predictions that might have harmful effects. Mathematically, precision is calculated as:

$$Precision = \frac{TP}{(TP + FP)}$$

- **Recall:** Recall, also known as Sensitivity or True Positive Rate, measures the model's ability to recognise actual positive cases properly. It comes in handy when False Negatives are a bigger problem than False Positives. The mathematical formula of the recall is presented as follows:

$$Recall = \frac{TP}{(TP + FN)}$$

- **F1 Score:** The F1 Score combines Precision and Recall into a single score, allowing for a more fair evaluation of a model's performance. It is particularly important when Precision and Recall must be balanced and their trade-off must be carefully considered. The F1 Score is the harmonic mean of Precision and Recall, and it is greatest when Precision and Recall are equal. Mathematically, F1 score is represented as follows:

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall}$$

## Chapter 5. Detailed Design and Implementation

This chapter digs into the deep specifics of the system's design and execution, giving light on every aspect that contributed to the final product. It will not only provide a full overview of the system's architecture and components, but it will also guide you through the use of specific technologies that helped shape the Food Recommender System. This chapter aims to equip readers with insights that facilitate the seamless continuation and evolution of the system in response to future demands by delving into the design principles and highlighting the implementation complexities.

### 5.1. Detailed Design

This section focusses on offering a better understanding of the Food Recommender System by illustrating not only the main architecture of the system, but also the use cases that highlight the need for this application.

#### 5.1.1. Main Architecture and Use Cases

As illustrates, our system's primary components are the following: Recommender System along with the dataset used, User Interface React Platform, MySQL Server and the Spring Boot Backend Server. Each of these key components have different functionalities and are interconnected.

- *Recommender System*: This primary component, implemented with Python 3, combines the functionalities and methodologies of both AHP, for classifying the appropriate foods, and KNN, the chosen machine learning algorithm that trains our system and seamlessly integrates the preferences into the recommendation process. With the help of python's capabilities, we are able here to efficiently process the data from the dataset into our menu generation algorithm.
- *User Interface React Platform*: This component represents our frontend application which provides, not only the gathering tool of critical information that our recommender system component needs, but also sets the communication means between our user (client or nutritionist) and our system.
- *MySQL Server*: The MySQL server is a critical component of our system, acting as a powerful database that stores and maintains the large amounts of data required by our Food Recommender System. Its sophisticated database capabilities assure the safe and orderly storage of user profiles, menu selections, and other pertinent data. Our solution smoothly obtains and changes information by using MySQL's dependability and scalability, allowing smooth interactions and improving the overall user experience.
- *Spring Boot Backend Server*: This component represents a critical part to our system, acting as a solid basis for data management and supporting smooth interactions. Spring Boot's extensive features allow it to handle user requests rapidly and process data from the MySQL database, for example preparing the data of the saved menus that a user has stored under his "id" in our database. This backend server provides a dependable and responsive user experience, and its modular architecture enables simple scaling and future expansions.

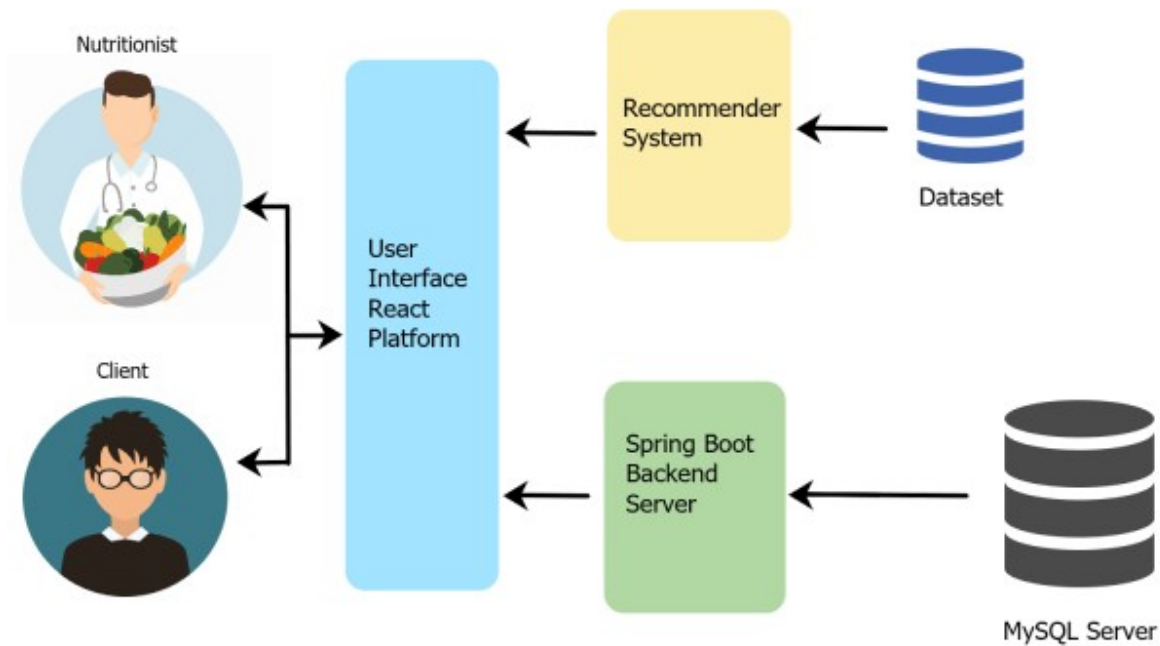


Figure 5.1 Main Architecture of the System

To further fortify our understanding of the architecture, we continue our discussion by providing some scenarios that will highlight the main functionalities of this license thesis. The following use cases will describe these scenarios by outlining how the application will react to certain real-world scenarios. For each of them we will present a short description of the scenario, a flow chart, the primary actor, the successful and alternative scenarios along with the preconditions and postconditions of the use case.

- *Generate Personalized Menu*

Description: Creating a personalized menu based on nutritional needs and user preferences. This use case presents how the goal of the system can be achieved (*Figure 5.2 Menu Generation Flow Chart*).

Primary Actor: Registered User

Successful Scenario:

1. The User logs into the application and enters the main page.
2. The User navigates to the “Generate menu form”.
3. The User enters their personal data, including dietary preferences.
4. The User submits the form.
5. The System validates the entered data.
6. The System processes the appropriate foods.
7. The System generates a menu by incorporating the user's preferences and fits it in the menu template.
8. The System presents the generated menu to the User.
9. The User chooses to either save the menu or generate a new one.

Alternative Scenario:

- Step 3a: If the user personal data is invalid, the system will display an error message and the user can modify his mistakes.

Preconditions:

- The user must be registered and authenticated.

Postconditions:

- The menu should stick to the menu template.

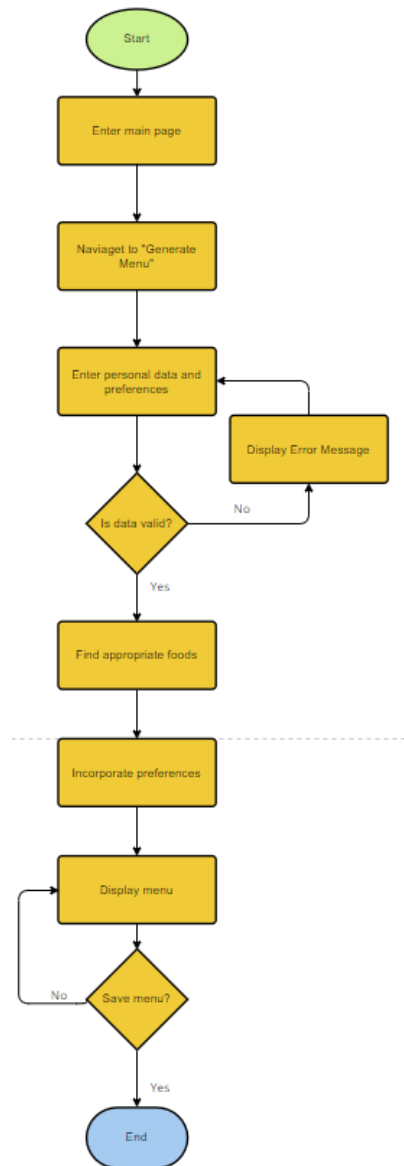


Figure 5.2 Menu Generation Flow Chart

- *View Menus and Add Items to the Shopping List*

Description: Accessing the user's menus and selecting the desired items to further have them in the shopping list ().

Primary Actor: Registered User

Successful Scenario:

1. The User logs into the application and enters the main page.
2. The User accesses the "View Saved Menus" section.
3. The System displays a list of previously generated or prescribed menus.
4. The User selects a specific menu to view its contents.
5. The System presents the menu details.
6. The User chooses certain foods from the menu and adds them to their shopping list.

7. The System displays the updated shopping list to the User.

Alternative Scenario:

- Step 4a: If the user decides to do another action and exists the menus page, the use case ends.

Preconditions:

- The user must be registered and authenticated.
- The user must have menus generated or prescribed.

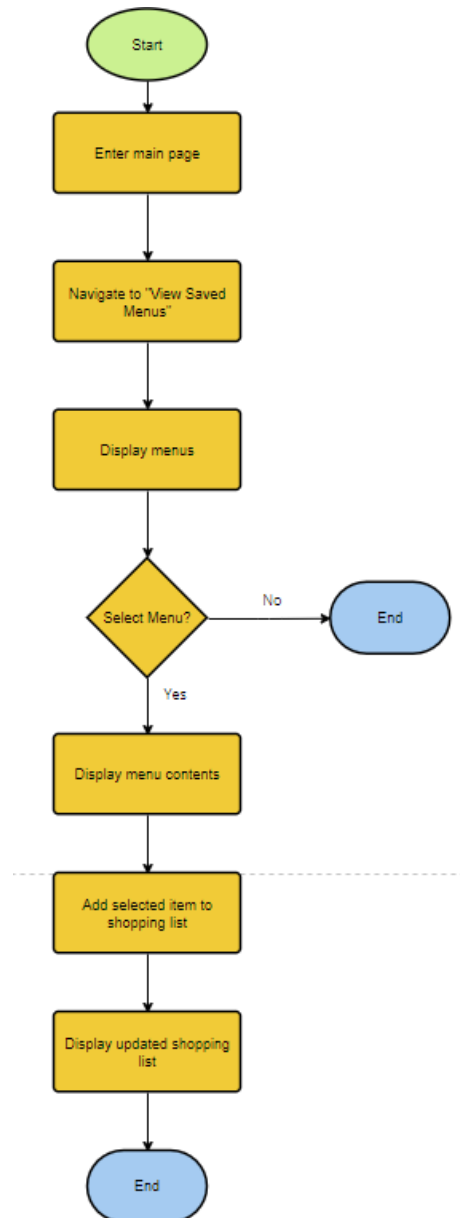


Figure 5.3 Add to Shopping List Flow Chart

• *Prescribe New Menu*

Description: The nutritionist prescribes a new menu, using their expertise and knowledge, for a user assigned to them (Figure 5.4 Prescribe Menu Flow Chart).

Primary Actor: Registered Nutritionist

Successful Scenario:

1. The Nutritionist logs into the application.

2. The Nutritionist checks their assigned user list.
3. The Nutritionist selects a user for whom they want to prescribe the menu.
4. The System will display the list of menus of the current user.
5. The Nutritionist navigates to the “Prescribe a new menu” form page.
6. The Nutritionist enters the foods along with their serving size and submits the new menu.
7. The System processes the new menu and saves it into the list of menus associated with the selected user.

Alternative Scenario:

- Step 2a: If the nutritionist has no users assigned to them, they are unable to select a user for menu prescription. The System displays an error message, and the Use Case ends.

Preconditions:

- The nutritionist must be registered and authenticated.

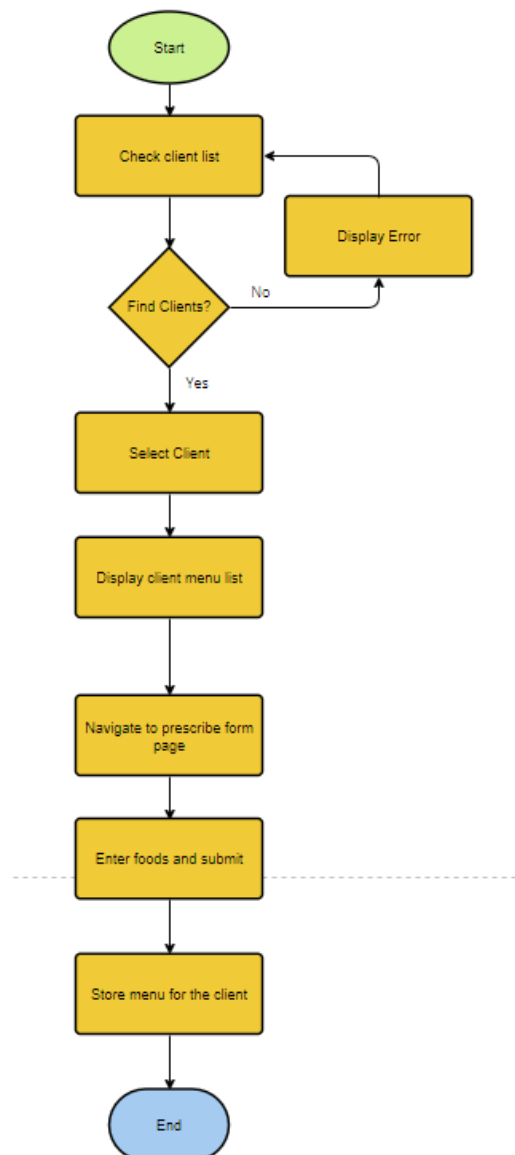


Figure 5.4 Prescribe Menu Flow Chart

## 5.1.2. Backend and Database Design

In the following section, we will dig into the complexities of our system's backend and database design models. This section will offer a detailed description of how the backend components are designed and managed to support our application's primary features. We'll also look at database architecture, showing how data is organised, stored, and accessed to guarantee maximum system efficiency and data integrity.

As mentioned above, the system's backend is provided by the Spring Boot framework. For a better understanding of the organization behind the written code, the package diagram () provides a graphical representation of the hierarchical structure of the backend application. It shows the key functional components and their relationships, offering an easy picture of how your application's many pieces interact and collaborate.

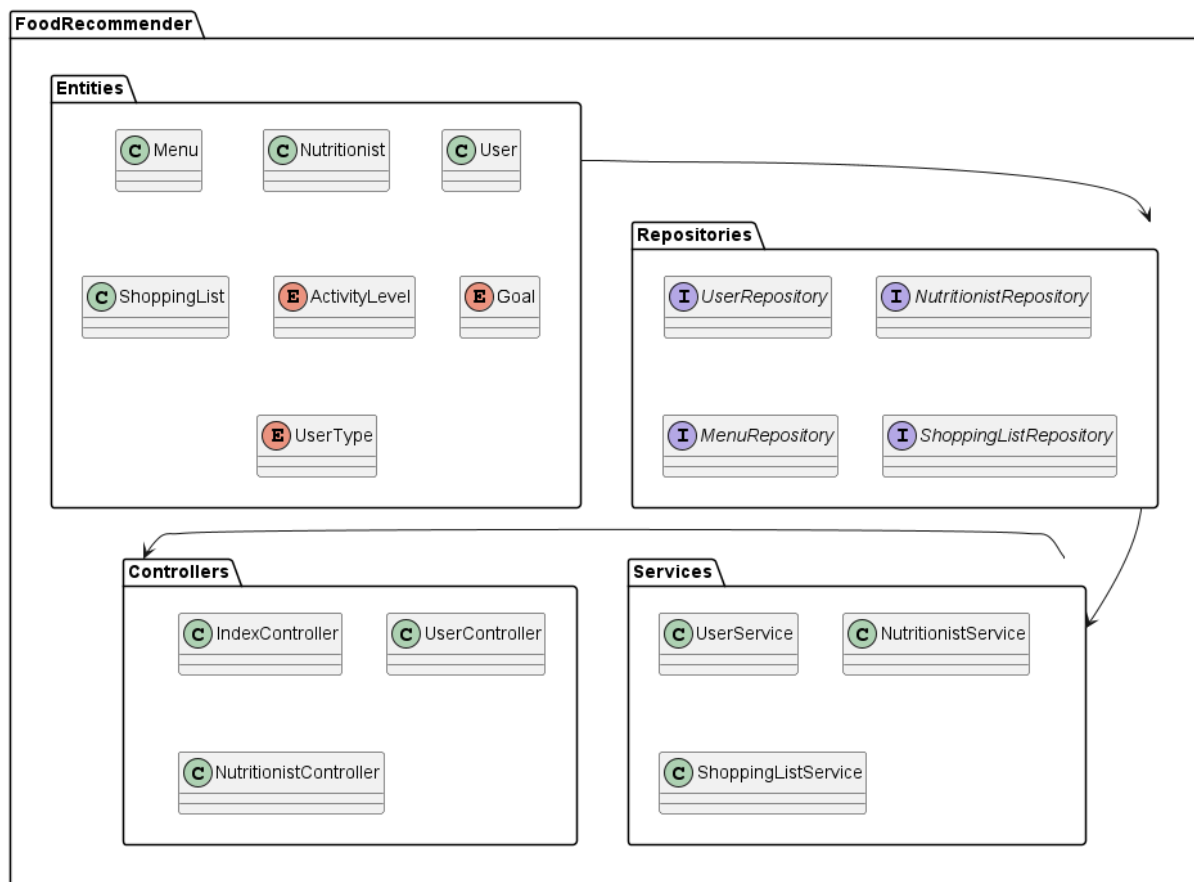


Figure 5.5 Package Diagram

As shown in the previous diagram, our backend Spring Boot application has a classic structure that follows a good modular and organized approach, having 4 main packages: Entities, Repositories, Controllers and Services. Each package represents a distinct set of capabilities, supporting a clear separation of responsibilities and a more manageable and scalable codebase.

- **Entities:** This package typically includes classes that reflect your application's data model. These classes describe the structure of your data entities, their properties, and their interdependence.
- **Repositories:** The Repositories package is responsible for data access and interaction logic. It includes interfaces or classes that extend Spring Data



repositories, allowing you to execute CRUD (Create, Read, Update, Delete) operations on databases.

- **Service:** The Service package includes business logic and services for interacting with repositories and performing sophisticated actions. These services serve as go-betweens for controllers and repositories, containing business rules and application-specific functionality.
- **Controllers:** The Controller package contains the API endpoints responsible for handling incoming requests and managing data flow between the client and the backend. Controllers take client input, call the necessary service methods, and provide replies.

Another good illustration of how our data is managed and preserved is represented by the database diagram. This diagram (*Figure 5.6 Database Diagram*) built using MySQL Workbench represents the underlying structure of our data and objects visually. This graphical model helps us to readily grasp and traverse the relationships between our database's tables, columns, and constraints. We can rapidly understand how data elements interact and are organized inside the system by visualizing the relationships and dependencies. This diagram is a useful tool for understanding the complexity of our data model, allowing us to find critical links and optimize the database architecture.

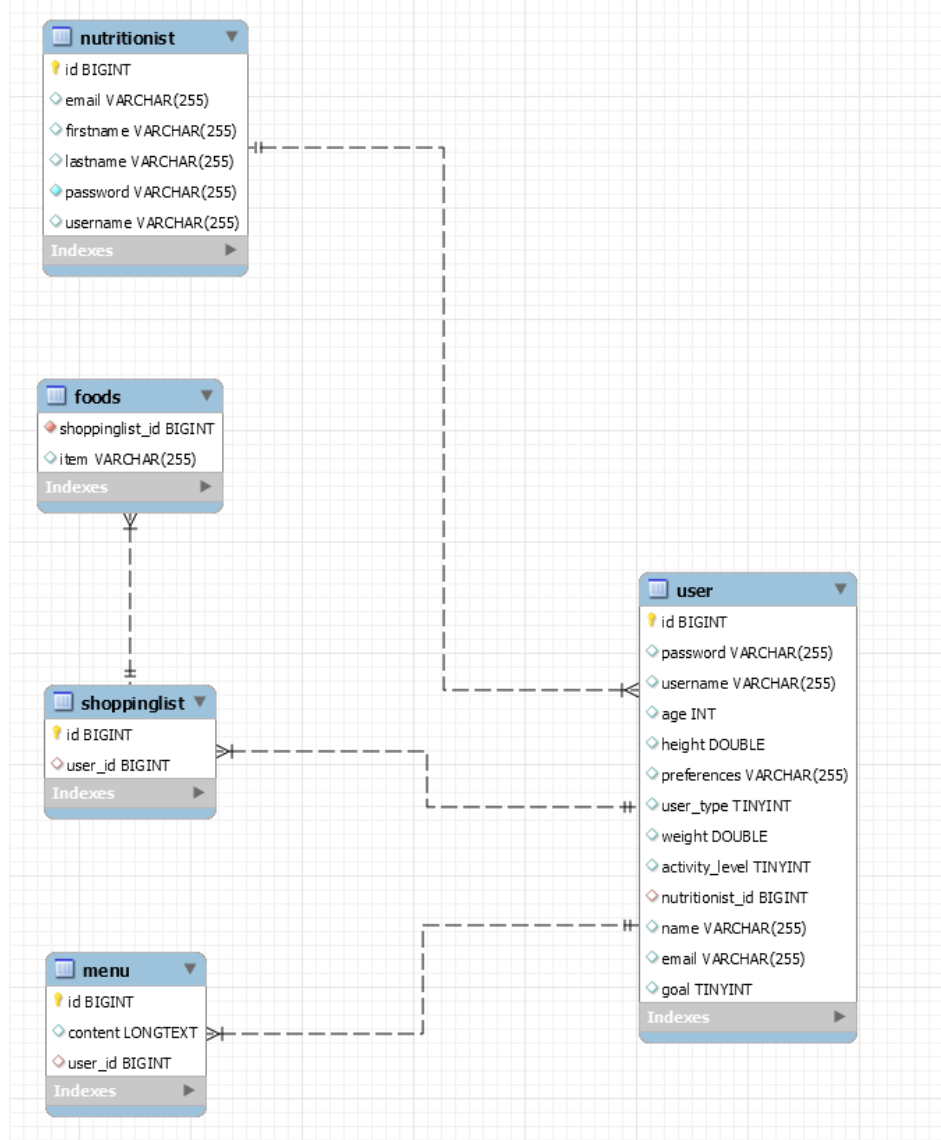


Figure 5.6 Database Diagram

### 5.1.3. Used Technologies and Frameworks

Before diving deep into the actual implementation of our Food Recommender System, I would like to mention some of the used technologies for the development of this application, but also highlight and describe their remarkable capabilities and features.

- **Spring Boot:**

Spring Boot is a popular Java-based framework meant to make developing web apps and microservices easier. Spring Boot, founded by Rod Johnson, provides a concise and opinionated method to addressing the difficulties commonly associated with setting up and configuring a Spring-based application. Spring Boot, a Spring ecosystem project, extends this approach by making it easier to create production-ready apps with minimum setup.

Spring Boot's ability to quickly set up and configure web applications, eliminating the need for manual configuration and boilerplate code, is one of its key features. It offers a complete set of tools and functions that significantly improve the development process. Spring Boot allows developers to concentrate on building application logic rather than dealing with complex setups.

Spring Boot provides several advantages in the context of web applications. Its built-in templates, quick configuration, and easy interface with multiple data sources, such as databases and APIs, enable speedy development. The usage of Java Persistence API (JPA) within Spring Boot simplifies database operations by making it simple to create, retrieve, edit, and remove records. Spring Boot also allows you to create executable JAR (Java Archive) or WAR (Web Archive) files, allowing you to deploy and run apps independently. The integrated web server capability eliminates the need for external web servers, simplifying deployment and testing. Spring Boot's strong support for third-party libraries and frameworks, along with its modular design, encourages a flexible and scalable development process.

Developers can significantly reduce the time and effort necessary to construct web apps by exploiting Spring Boot's features. The design concepts and feature set of the framework cater to different areas of current web development, from database connectivity to security implementation. Spring Boot, in essence, enables developers to construct strong, scalable, and efficient online applications without being weighed down by complex setups and repeated processes.

Because of its exceptional capabilities in simplifying web development while focusing on key backend activities, Spring Boot is an excellent choice for our Food Recommender System web application. As previously stated, Spring Boot's convention-based approach and fast setup are especially beneficial in avoiding boilerplate code and quickly establishing a powerful backend architecture. This nicely matches with the aims of our project, allowing us to focus on developing an efficient recommendation engine and user-friendly interfaces.

- **React Framework:**

React is a popular and famous JavaScript package used to create user interfaces for online applications. React was built by Facebook to address the issues of efficiently handling dynamic and sophisticated UI components in modern web development. By providing a component-based design and pushing the notion of declarative user interfaces, this toolkit has transformed the way developers approach frontend development.

The ability to construct reusable and encapsulated UI components is React's core strength. These components, each representing a different aspect of the user

interface, may be combined to create advanced and elegant UIs. Developers may manage the codebase more effectively, increase code reusability, and simplify maintenance by using a component-based approach.

Declarative user interfaces are a key component of React. Developers use this method to specify how the UI should appear at any given moment, and React handles updating the UI to reflect the intended state. This idea streamlines the administration of UI changes, making it easier to track and maintain the visual appearance of the system.

Another important aspect that adds to React's efficiency and speed is its virtual DOM (Document Object Model). The virtual DOM is a lightweight approximation of the actual DOM that allows React to update and render changes while avoiding direct manipulation of the real DOM. This method eliminates the overhead associated with frequent changes, resulting in speedier and more consistent user experiences. React also enables one-way data flow, which ensures that changes in a component's state prompt UI updates. This unidirectional data flow facilitates debugging and aids in the maintenance of a predictable state management system.

React's ability to generate interactive and responsive user interfaces is useful in the context of web applications. It interfaces effortlessly with an array of frontend libraries and state management tools, making it a versatile solution for developing single-page and multi-page apps.

Developers may increase their productivity by reusing components, increasing code organisation, and taking use of React's speed optimisations. Because of React's accessibility and strong community support, a significant ecosystem of tools, frameworks, and resources that further ease the development process has emerged. React helps developers to construct extremely efficient, maintainable, and engaging online apps, whether they are designing simple UI components or sophisticated user interfaces. The articles [\[23\]](#) and [\[24\]](#) also offer an insightful and easy to understand introduction to how to use this technology.

- **MySQL:**

MySQL is an open-source relational database management system (RDBMS) that is essential for managing and organising structured data in a variety of applications. It was created by MySQL AB and is currently owned by Oracle Corporation. MySQL is well-known for its robustness, versatility, and ease of use, making it one of the world's most popular database systems.

MySQL is built on the relational database paradigm, which stores data in tables with rows and columns. This structure facilitates data retrieval, manipulation, and organisation. MySQL interacts with the database using Structured Query Language (SQL), which provides a standardised and sophisticated means to execute operations such as querying, adding, updating, and removing data.

Scalability is one of MySQL's primary features. It is capable of handling both small-scale applications and large-scale business solutions. MySQL, with its capacity to store terabytes of data, is an excellent choice for applications with a wide range of data storage requirements.

MySQL provides a number of storage engines that govern how data is saved and retrieved. For example, the InnoDB storage engine has capabilities like as transactions and foreign key restrictions, making it suited for applications requiring high data integrity. Another storage engine that provides good performance for read-heavy workloads is MyISAM.

MySQL is frequently used in online applications to store and manage data about users, goods, transactions, and other topics. It works well with a variety of programming languages and frameworks, including Spring Boot, making it a popular choice for backend data storage.

MySQL's ability to provide ACID (Atomicity, Consistency, Isolation, Durability) transactions is one of its fundamental characteristics, guaranteeing that database operations are dependable and data integrity is maintained. This is especially significant in applications that rely on data consistency.

To boost query performance, MySQL also supports a variety of data formats, indexing systems, and query optimisation techniques. Because it supports sophisticated features including as views, stored procedures, triggers, and events, developers may construct complicated database logic directly within the database system.

- **Python libraries:**

Python is a popular and flexible programming language that is noted for its readability and simplicity. It has a large ecosystem of libraries and frameworks that cater to a wide range of fields, making it a popular choice among developers for web development, data analysis, machine learning, and other tasks. Here are some libraries that made an impact in developing this recommender system:

- **Pandas:**

As presented in the article [\[25\]](#), Pandas is a Python data manipulation and analysis toolkit. It includes data structures and operations that allow for the effective management and transformation of structured data, making it essential for activities such as cleaning, converting, and analysing information. In the context of this license thesis, especially in the recommender system component, Pandas became usefull for operations like importing and preprocessing data.

- **NumPy:**

NumPy is a Python package that is required for scientific computing and data handling. It teaches strong data structures like arrays and matrices that serve as the foundation for executing mathematical operations effectively. This is very important in machine learning and data analysis jobs. NumPy's array operations are optimised, allowing it to do large-scale numerical computations much quicker than regular Python lists. NumPy can handle data preparation, modification, and even complicated mathematical computations required by algorithms such as AHP and KNN.

- **Sklearn (scikit-learn):**

Scikit-learn, sometimes known as sklearn, is a comprehensive machine learning package that makes it easier to implement numerous algorithms. Its user-friendly architecture gives developers access to a diverse set of machine learning models for classification, regression, clustering, and other tasks. Scikit-learn additionally includes data preparation, feature selection, and model assessment tools. In the context of your project, sklearn played a major role in the implementation of the KNN method and other machine learning features. Its uniform API structure and rich documentation allow experimenting with various algorithms and strategies

easy. The paper [26] offers a better understanding of this library along with the code desing and underlying technologies.

- Flask:

Flask is a micro web framework that allows developers to construct web apps that are both simple and flexible. Flask provides the fundamental tools required to create web services and APIs. It's ideal for small to medium-sized projects when a full-fledged framework could be too much. Flask's modular design allows you to connect different extensions as required, making it extremely configurable. Flask was a key component for this system by creating a way of comunication between the frontend and our recommender application.

## 5.2. Implementation

In the upcoming section, we will take a deeper look into the practical coding components of our license thesis. The conceptual design and planning start to take shape as real code at this point. We will investigate the process of turning our system's architecture, concepts, and technologies into functional components step by step, also walk through the design of crucial modules, their linkages, and how they together bring our Food Recommender System to life using extensive explanations and code samples. This area allows you to see the physical manifestation of our thoughts and concepts as we go on a path of hands-on coding and development of each key component.

### 5.2.1. The Recommender System

The Recommender System application does not only represent the heart of our license thesis, but also what makes our web application unique. The main task of this component is to seamlessly gather all the user information needed, along with the preferences, and generate, by processing the food items provided by our dataset, a healthy and personalized meal plan for the current day. This application was developed using Visual Studio Code and implemented in Python, along with the needed libraries (), some of which were already mentioned in the previous sections.

```
import pandas as pd
import math
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from flask import *
from flask_cors import CORS
import json
import random
```

Figure 5.7 Imported libraries of the system

We will now analyze the code for each function inside the application in detail, explaining their respective purposes and functionalities:

- “*process\_alternative\_csv(input\_csv)*”:

The main goal of this function () is to gather and preprocess the collection of foods from the dataset. Before the function's declaration we first create a new instance of a new Flask web application. The next line configures CORS for the Flask app. CORS is a web browser security feature that prevents unauthorized domains from making requests to another site. By enabling CORS, you select which sources are permitted to access your Flask app. Only requests from "http://localhost:3000" are permitted in this scenario. Furthermore, you permit these origins to transmit headers with the identifier "Content-Type". The function only takes a single argument "input\_csv", which should be represented by the path of the CSV file of the dataset. Next, we read the file and save it into a Pandas data frame named "alternative" and use the argument "header=0" to indicate that the first row of the dataset contains the names of each column. After that we fill every empty space of the data frame with 0 and after dropping some columns to improve our machine learning algorithm, we change every 0 to a very small value so that we can compute in the following functions the AHP algorithm. Lastly the processed data frame is returned.

```
app = Flask(__name__)
CORS(app, origins='http://localhost:3000', allow_headers=['Content-Type'])

def process_alternative_csv(input_csv):

    alternative = pd.read_csv(input_csv, header=0)
    alternative.fillna(0, inplace=True)
    alternative.drop(['Calcium', 'Iron', 'Potassium', 'VitaminC', 'VitaminE', 'VitaminD'], axis=1, inplace=True)
    alternative = alternative.replace(0, 0.000001)
    return alternative
```

Figure 5.8 Preprocessing the data.

- "calculate\_bmr(weight, height, age)":

This function (Figure 5.9 BMR, BMI and calorie computation functions) requires as parameters, the weight (kg), the height (cm) and the age (in years) of a user in order to compute his Basal Metabolic Rate (BMR) and Body Mass Index (BMI) and returns them. The method uses the Harris-Benedict equation in order to compute the BMR which is shown in the first line of the function and then calculates the BMI by dividing the weight of the user with the height squared in meters.

- "calculate\_final\_calories(weight, height, age, activity\_level, goal)":

The main task of this function is to return the final caloric need of the user, based on the arguments that are given. The method takes the arguments: weight, height and age to compute the BMR and BMI by calling the previous function. The other 2 arguments represent, as their names suggest, the activity level and dietary goal of one's user. We also create a table, named "activity\_level\_table" that will hold the levels of activity, which are also similar with the ones from the database, and the BMR factor, which are multipliers used to alter BMR based on an individual's level of activity. The next step is represented by the calorie calculation, which revolves around adjusting the BMR according to the "activity\_level" and then we round up the result to the nearest multiple of 10 using the "math.ceil" function.

Lastly, we adjust the calorie value depending on the "goal":

- If the target goal is "GAIN", the calorie value will be increased by 300.
- If the target goal is "LOSE", the calorie value will be decreased by 300.
- Otherwise, the value stays the same.



```

def calculate_bmr(weight,height,age):
    my_bmr = 66 + 13.75 * weight + 5 * height - 6.75 * age
    bmi = weight/ (height*height/100)
    return my_bmr, bmi

def calculate_final_calories(weight, height, age, activity_level,goal):
    my_bmr, bmi = calculate_bmr(weight,height,age)
    activity_level_table = {
        'Activity Level': ['LITTLE', 'LIGHT', 'MODERATE', 'STRONG', 'EXTREME'],
        'BMR Factor': [1.2, 1.375, 1.55, 1.725, 1.9]
    }
    activity_index = activity_level_table['Activity Level'].index(activity_level)
    my_calories = my_bmr * activity_level_table['BMR Factor'][activity_index]
    new_calories = math.ceil(my_calories / 10) * 10
    if goal == 'GAIN':
        final_calories = new_calories + 300
    elif goal == 'LOSE':
        final_calories = new_calories - 300
    else:
        final_calories = new_calories
    return final_calories

```

Figure 5.9 BMR, BMI and calorie computation functions.

- “get\_limiting\_profile(user\_type, bmi)”:

The function's goal () is to offer nutritional restrictions based on the user type and BMI. It accomplishes this by referencing a predetermined profile of nutritional limitations for several user types and selecting the most restrictive restriction for each nutrient between the specified user type and a type indicated by BMI. The arguments of the function are “user\_type”, which represents the health category of the user (“HEALTHY”, “DIABETES” and “HYPERTENSIVE”), and “bmi” which represents the Body Mass Index of the user. With the help of the second parameter, the function can combine the health category of the user with the ones that are resulted from the BMI value, which can be “OVERWEIGHT”, “UNDERWEIGHT” or it should stay “HEALTHY”.

We use the dictionary “limiting\_profile” to define the nutritional limitations for a variety of health conditions. Each nutrient has a set of values that match to the user kinds given in “UserType”. After that, based on the BMI value, this algorithm determines the user type. The user is classified as “UNDERWEIGHT” if their BMI is less than 20. They are classified as “OVERWEIGHT” if their BMI is greater than 25. Otherwise, they are classified as “HEALTHY”, then stores it in the variable “userT”. The nutrient limits for the provided “user\_type” and the BMI-determined “userT” are extracted from the predefined limits. The “UserType” column is then dropped, as it's no longer needed. Because there are two sets of limitations (one for “user\_type” and one for “userT”), the function takes the minimum value between the two to establish the most restrictive limit for each nutrient and then the function returns the final values.

```
def get_limiting_profile(user_type, bmi):
    limiting_profile = {
        'Protein': [100, 120, 100, 100, 120],
        'TotalFat': [80, 72, 80, 80, 80],
        'Carbohydrate': [240, 216, 192, 240, 288],
        'Sodium': [3000, 3000, 3000, 1500, 3000],
        'SaturatedFat': [60, 54, 60, 60, 60],
        'Cholesterol': [300, 300, 300, 300, 300],
        'Sugar': [70, 63, 56, 70, 70],
        'UserType': ['HEALTHY', 'OVERWEIGHT', 'DIABETES', 'HYPERTENSIVE', 'UNDERWEIGHT']
    }

    userT = 'HEALTHY'
    if bmi < 20:
        userT = 'UNDERWEIGHT'
    if bmi > 25:
        userT = 'OVERWEIGHT'

    limiting_profile_df = pd.DataFrame(limiting_profile)
    result = limiting_profile_df[limiting_profile_df['UserType'].isin([user_type, userT])]
    result = result.drop(['UserType'], axis=1)
    values = result.min().values
    return values

### Comparison
```

Figure 5.10 Computing the limiting profiles of each function.

- “*create\_comparison\_matrix()*”:

The main purpose of this function is to create and return the comparison matrix for all the remaining macronutrients and micronutrients that belong to each food’s composition. This matrix (), representing the priorities between each nutrient, is required in the AHP classification algorithm in order to determine which foods are appropriate for our user.

- “*calculate\_weight\_from\_matrix(cmp\_matrix)*”:

This function is designed to compute weights or priorities from a pairwise comparison matrix, which is an essential component of the Analytic Hierarchy Process (AHP) approach. AHP is a decision-making approach that provides for a rank-based multi-criteria analysis procedure. First, the input matrix “*cmp\_matrix*” is converted into a NumPy array with data type set as float and then we take its transpose and normalize it ensuring that the sum of each column is 1. After that we compute the eigenvalues and eigenvectors of the resulted matrix, using NumPy’s linear algebra. Before computing the weights vector, the function verifies if the comparison matrix is consistent or inconsistent by computing the consistency ratio “CR” and then compare it with 0.1. As a general rule, if CR is less than 0.1, the matrix is considered consistent. If it is more, the matrix is incoherent, and the pairwise comparison judgements may need to be revised. Finally, by normalizing the primary eigenvector, the weights are obtained. The resulting weights equal one. For clarity, the weights are then rounded to three decimal places.



	Protein	TotalFat	Carbohydrate	Sodium	SaturatedFat	Cholesterol	Sugar
Protein	1	0.333333	0.5	0.5	0.333333	0.5	0.333333
TotalFat	3	1	3.0	3.0	2.0	2.0	2.0
Carbohydrate	2	0.333333	1	2.0	0.5	1.0	0.5
Sodium	2	0.333333	0.5	1	0.5	1.0	1.0
SaturatedFat	3	0.5	2	2	1	2.0	2.0
Cholesterol	2	0.5	1	1	0.5	1	1.0
Sugar	3	0.5	2	1	0.5	1	1

Figure 5.11 A look at the resulted comparison matrix of the nutrients.

- “*calculate\_appropriate\_foods(limiting\_profile, alternative, weights)*”:

This function () determines which foods from the dataset are suitable based on the particular nutrient limitation profile “limiting\_profile” and nutrient “weights”. This is accomplished by comparing the nutritional profiles of each item to the limiting profile using a pairwise comparison matrix for each nutrient, then evaluating these comparisons depending on the nutrient weights supplied, also following the AHP algorithm mentioned in the previous chapters.

First, we convert the “limiting\_profile” to a list and we create an empty data frame with the same columns as “alternative”. Next, by iterating over all the food entries, we take each of the 6 nutrients and make a pairwise comparison matrix “M” of 2x2. This matrix compares the nutrient value in the “limiting\_profile” to the nutrient value in the current food from the alternative dataset. After that, we determine the local priorities, by finding the primary eigenvector and normalizing it. These priorities show how the nutritional value of the present food relates to the nutrient value of the limiting profile. Then we weight these local priorities using the nutritional weights supplied to give importance to some nutrients over others. As the AHP algorithm suggest in the last steps, we sum up the weighted local priorities of the food “suma” and of the limiting profile “suml”. If the food's weighted sum is less than or equal to the limiting profile's weighted sum, the food is judged suitable and is added to the appropriate data frame. In the end the function returns the “appropriate” data frame.

- “*classify\_appropriate\_foods(appropriate)*”:

This method aims to classify the appropriate foods based on their IDs into different food types. First, we convert the data types of each column in order to have them correctly set, even the manually added column “ServingSize”, which, as the name suggests, represents the serving size in grams of each food. This classification is needed in order to improve our response time of the system and to accurately keep our foods in classes that will help us fit them later in the menu template. The next lines show the classes and the ID ranges related to them.

- Milks: 1000 – 1255.
- Proteins: 5000 – 5677, 7000 – 7960, 9999 – 10993, 12999 – 13987, 15000 – 15265, 17000 – 17348, 22999 – 23660.
- Cereals: 8000 – 8644.
- Fruits: 9000 – 9452.
- Veggies: 11000 – 11998.
- Carbs: 16000 – 16386, 18000 – 18982, 20000 – 20649.
- Others: every other ID.

After this classification we are removing the “Other” column, that is mostly composed of oils, condiments, and drinks, and we return the updated “appropriate” data frame.

```
def calculate_appropriate_foods(limiting_profile, alternative, weights):
    limiting_list = limiting_profile.tolist()
    appropriate = pd.DataFrame(columns=['ID', 'Description', 'Calories', 'Protein', 'TotalFat', 'Carbohydrate',
                                       'Sodium', 'SaturatedFat', 'Cholesterol', 'Sugar', 'ServingSize'])
    poz = 0

    for v in range(len(alternative)):
        local_list_limiting = []
        local_list_alternatives = []
        for i in range(0, 6):
            M = np.zeros((2, 2))
            M[0, 0] = 1
            M[0, 1] = abs(limiting_list[i] / alternative.iloc[v][i + 3])
            M[1, 0] = abs(alternative.iloc[v][i + 3] / limiting_list[i])
            M[1, 1] = 1
            egnval, egnvect = np.linalg.eig(M)
            maxegnval = np.argmax(egnval)
            domegnvector = egnvect[:, maxegnval]
            local_priorities = domegnvector / domegnvector.sum()
            local_prio = [round(value, 4) for value in local_priorities]
            local_list_limiting.append(local_prio[0] * weights[i] / 6)
            local_list_alternatives.append(local_prio[1] * weights[i])
        lla = [round(value, 4) for value in local_list_alternatives]
        lll = [round(value, 4) for value in local_list_limiting]
        suma = np.sum(lla) / 7
        suml = np.sum(lll) / 7

        if suma <= suml:
            appropriate.loc[poz] = alternative.iloc[v]
            poz += 1

    return appropriate
```

Figure 5.12 AHP Classification.

- “train\_knn\_model(appropriate)”:

This method () is designed to train the KNN classifier using the “appropriate” data frame given as argument. The function scales the data, divides it into training and testing sets, trains a k-NN model, and returns the trained model together with the testing set. The independent variables (features) are extracted from the appropriate data frame by removing columns such as “ID”, “Description”, “FoodType”, and “ServingSize”. The result saved in “X” and after we rename the columns for clarity. We also define the dependent variable (target) and store it in “y”, which is the “Description” column. After that we standardize the features using the “standardization(feature)” function. Before training our classifier, the standardized data is divided into two sets: training and testing. 80% of the data is utilized for training, with the remaining 20% for testing. To ensure consistency, the “random\_state” parameter is set to 50. For the training process, I chose to best fit our system that the KNN should be trained with only one neighbor “k=1”. Finally, we return the trained model “knn”, the feature matrix of the testing set “X\_test” and the target values of the testing set “y\_test”.

```
def train_knn_model(appropriate):
    X = appropriate.drop(['ID', 'Description', 'FoodType', 'ServingSize'], axis=1)
    X.columns = ['Calories', 'Protein', 'TotalFat', 'Carbohydrate', 'Sodium', 'SaturatedFat', 'Cholesterol', 'Sugar']
    y = appropriate['Description']
    X_scaled = X.apply(standardization)
    X_train, X_test, y_train, y_test = train_test_split(X_scaled, y.values.ravel(), test_size=0.2, random_state=50)
    knn = KNeighborsClassifier(n_neighbors=1)
    knn.fit(X_train, y_train)
    return knn, X_test, y_test
```

Figure 5.13 Training of the KNN Model.

- “`find_closest_food(preferences, appropriate, knn, X_test, y_test)`”:

The function (Figure 5.14 Finding the closest food for recommendation.) was created to locate the closest matching food from the appropriate data frame depending on a user's preferences. It accomplishes this by first locating the most comparable meal description in the testing set, using the “`similar_food(s1,s2)`” that takes 2 strings and returns the similarity score, and then utilizing the KNN model to predict a new food that will fit the preferences of the user, based on the feature values of that food.

The method starts with computing the similarity scores between the “preferences” given as argument and every food description from the testing set “`y_test`”. On the next 2 lines, the function finds the higher index from the similarity scores resulted before, to use it in predicting the next food and save it in the “`closest_foods`”. Lastly, we use the predicted description to return extracted food from the “`appropriate`” data frame.

```
def find_closest_food(preferences, appropriate, knn, X_test, y_test):
    sim_scores = [similar_food(preferences, desc) for desc in y_test]
    max_index = max(range(len(sim_scores)), key = sim_scores.__getitem__)
    closest_foods = knn.predict([X_test.iloc[max_index]])
    food = appropriate[appropriate['Description']==closest_foods[0]]
    return food
```

Figure 5.14 Finding the closest food for recommendation.

- “`generate_menu(appropriate, preferences, calories)`”:

This method, represents the whole process of our menu creation and how every food will be added to the final meal plan, according to their category. The function intends to generate a menu based on the user preferences and calorie objective. The menu is built using the “`appropriate`” data frame, ensuring that the overall calorie count of the chosen food products does not exceed the calorie objective. To further break down this method, it begins by training the model using the aforementioned function “`train_knn_model()`” and initializing the lists for storing our foods. Next, for each word in the “`preferences`”, the closest matching food is determined and categorized based on its type. The function then starts a loop that runs until the total calories of the menu items picked surpass the goal calorie count. If a food is on the user preference list for each food category (Milks, Cereals, etc.), it is added to the menu. If not, a random meal is chosen and inserted from the “`appropriate`” data frame for that category.

The loop guarantees that the menu includes the required food types of the menu template (). If a category has several preferences, the function takes into account up to two entries from that category. Before returning the created menu “`foods`”, the function also computes the evaluation metrics based on the KNN model, which we will dive into in the next chapter, and returns them.

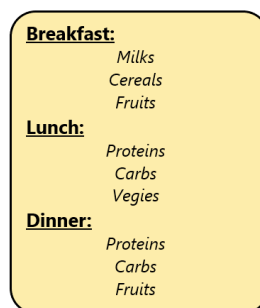


Figure 5.15 Menu Template.

- “*recommended\_menu()*”:

The last part of our recommender system is represented by this function. Here we put together the resources, tools and methods implemented above to make our meal generation complete. But before that, we are setting up our Flask web server with a single route (“*receive\_user*”) that listens for POST and OPTIONS requests. The function generates a default options response if the request method is OPTIONS. This is most likely for dealing with CORS preflight requests. If the request method is POST, the server expects to receive JSON data with user details and preferences. This function prepares the data, by using the aforementioned functions and collects the dataset from the hardcoded path of the CSV file, to be sent to our user interface, by converting it to JSON. After that, the last lines of the application (*Figure 5.16 Application start.*) make our system run properly on port 7777.

```
if __name__ == '__main__':  
    app.run(port=7777)
```

*Figure 5.16 Application start.*

### 5.2.2. Backend App

The second section delves into the core components of our implementation, concentrating on the complexities of our Spring Boot backend application. The combination of the Java programming language and IntelliJ IDEA provides the necessary tools and environment for the smooth creation of this backend application, providing efficient coding, debugging, and component integration. Our backend application's major function is to facilitate smooth communication between the front-end user interface and the database. The application itself serves as the engine that drives our whole system, allowing users to conduct a variety of critical tasks. One of its primary roles is to perform CRUD (Create, Read, Update, Delete) actions on the database, which ensures that data is maintained effectively and properly, but before digging deeper into the implamantation of these operations, we should take a look and understand the connection process between our database and backend.

One of the key components in setting up our backend project is the “pom.xml” file. This file defines the project dependencies, build settings, create our Spring Boot application and other metadata. For the connection of our project with our MySQL database we have defined the MySQL Connector dependency that makes possible this connection by adding the MySQL JDBC driver. Along with this dependency, multiple other dependencies are occupying the “pom.xml” file, but we can also mention the H2 Database dependency. With this dependency we establish an H2 in-memory database that doesn’t require installation and plays a crucial role in selecting which dialect we use for generating SQL queries.

The last step in completing our database connection is done with the aid of the “application.properties” file (), which holds our database connection properties. In this file we specify the JDBC URL for connecting to our local MySQL database with the name “*food\_recommender\_system*”. We define here the username and password that are required for connecting our application to our database and we also specify, with the line “*spring.jpa.hibernate.ddl-auto = update*”, that Hibernate should update our database at the start of the backend app, if necessary.

```
spring.datasource.url=jdbc:mysql://localhost:3306/food_recommender_system?useSSL=false
spring.datasource.username=root
spring.datasource.password=pass

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect

spring.jpa.hibernate.ddl-auto = update
```

Figure 5.17 Contents of “application.properties”.

As mentioned before in the *Detailed Design* section, our backend development revolves around 4 main packages: Entities, Repositories, Service and Controllers ().

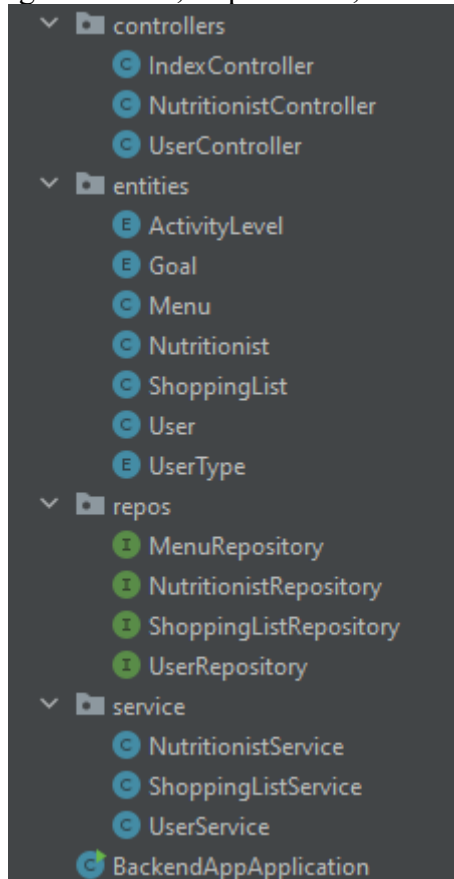


Figure 5.18 Package structure.

We will begin our exploration of the implementation by breaking down each one of these packages:

- Entities:

This package forms the foundation of our application's data model. It contains the core data structures and represents the real-world things and concepts with which our system interacts. These entities represent the domain's basic components, such as users, menu items, preferences, and other relevant aspects. A blueprint that outlines the properties and behaviors of a certain type of object is what an entity is. These entities are utilized in the context of a backend application to map business entities to related database tables. This procedure is known as Object-Relational Mapping (ORM).

We build a bridge between the logical world of our application and the physical world of the database by carefully constructing entities that appropriately reflect our system's domain. Entities define the data structure by describing attributes, relationships, and

constraints. We provide a clear correlation between entity properties and database columns using annotations or settings, assuring data integrity and consistency.

We will now look at the contents of one of the main classes of this package: User (Figure 5.19 User Class definition.). As the name suggests, the user class represents the user/client in the system. It is using the “@Entity” annotation to indicate that it is a JPA entity and that it will be mapped to our database table. This class has the following attributes:

- “id”: This is the user entity's primary key, which is produced automatically (“@GeneratedValue” with “GenerationType.IDENTITY”).
- “username”: Represents the unique username of the user.
- “password”: Represents the password of the user.
- “name”, “email”, “height”, “age”, “weight”: Represents the personal details of the user which are needed for the menu generation algorithm.
- “userType”: Using an enum this field represents the health status of the user.
- “preferences”: Is a string that contains the preferences of an user.
- “activityLevel”: Using an enum this field represents the activity level an user has.
- “goal”: Using an enum this field represents the weight goal an user has.
- “nutritionist”: Represents a many-to-one relationship with the Nutritionist entity, holding the nutritionist information that the user is assigned to.
- “shoppingList”: Represents a one-to-one relationship with the ShoppingList entity.

Along with this fields, our User class has 3 constructors: a full constructor with all the attributes, an empty one and a constructor that only initializes the “id”, “username” and “password”, mostly used for the authentication operation. The entity also has generated the classic getters and setters methods, along with the “equals()”, “hashCode()” and “toString()”, useful for future operations or debugging.

```

8  @Entity
9  public class User {
10     @Id
11     @GeneratedValue(strategy = GenerationType.IDENTITY)
12     private Long id;
13
14     @Column(name="username", unique = true)
15     private String username;
16
17     @Column(name="password", nullable = false)
18     private String password;
19
20     @Column(name="name")
21     private String name;
22
23     @Column(name="email")
24     private String email;
25
26     @Column(name="height")
27     private Double height;
28
29     @Column(name="weight")
30     private Double weight;
31
32     @Column(name="age")
33     private Integer age;
34
35     @Column(name="userType")
36     private UserType userType;
37
38     @Column(name="preferences")
39     private String preferences;
40
41     @Column(name="activityLevel")
42     private ActivityLevel activityLevel;
43
44     @Column(name="goal")
45     private Goal goal;
46
47     @ManyToOne
48     @JoinColumn(name="nutritionist_id")
49     private Nutritionist nutritionist;
50

```

Figure 5.19 User Class definition.

The other entities of this package are implemented in the same fashion, besides the enums. All of them are using the same “@Entity” annotation and “@Column” annotation to correctly map them to our database. There are also other annotations that are signaling the use of relationships, such as “@ManyToOne” and “@JoinColumn”.



- Repositories:

Moving on to the "Repositories" package, we come across an important component that links our backend application to the database utilizing the power of JPA (Java Persistence API) repositories. This package is critical for controlling the persistence of our entities in the database. The main objective is to abstract away the complicated database interactions, enabling us to interface with the database using simple and straightforward methods offered by JPA repositories. These repositories function as interfaces, defining a collection of standard methods for performing typical database operations like creating, reading, updating, and deleting records (CRUD operations). We inherit these functions via extending JPA repository interfaces rather than implementing the methods ourselves.

This method eliminates the need to create specific SQL queries for common CRUD tasks, resulting in a cleaner, more manageable, and error-free codebase. JPA repositories produce SQL queries based on method names, freeing developers from the complexities of database operations.

The "Repositories" package and the "Entities" package have a close connection. Each repository corresponds to a single object type. The repository methods' structure matches the properties and relationships declared within the entity. This alignment guarantees that the repository methods naturally fit with the associated entity's business logic and use cases.

This package acts as a bridge between the business logic of our application and the database. We leverage the power of abstraction and automation by employing JPA repositories, simplifying database interactions while maintaining data integrity and consistency. This package converts our conceptual understanding of data manipulation into real, easily available techniques that can be smoothly implemented into the services and controllers of our application.

A good example of the use of the JPA repository is illustrated by , where we can see the whole implementation of our MenuRepository interface.

```

1 package com.diplomaproject.backendapp.repos;
2
3 import com.diplomaproject.backendapp.entities.Menu;
4 import com.diplomaproject.backendapp.entities.User;
5 import org.springframework.data.jpa.repository.JpaRepository;
6
7 import java.util.List;
8
9 public interface MenuRepository extends JpaRepository<Menu, Long> {
10     List<Menu> findByUser(User user);
11
12 }

```

Figure 5.20 MenuRepository definition.

- Service:

Moving on to the "Services" package, we explore the core of our backend project, which houses complex business logic. This package is in charge of organizing and implementing our application's fundamental functionality. We mark these classes as service components that contain the application's critical functions using Spring's "@Service" annotation. This means that Spring will automatically register and detect this class as a bean during the component scanning process.

The "Services" package's major goal is to encapsulate and abstract business functionality away from controllers and repositories, guaranteeing a clear separation of concerns and fostering code modularity. This design decision improves code organization, maintenance, and readability.

We develop and implement methods in each service class that encapsulate specific actions or workflows required by our application. These approaches use a combination of repository operations and extra logic to perform tasks ranging from data processing to complicated algorithmic computations. Let's take a look now at the UserService class ().

As already mentioned, the class starts with the "@Service" annotation, that specifies that this class is a service component of our Spring application. The attributes of this class are:

- "userRepository": Represents an instance of UserRepository that will provide CRUD operations for the User entity.
- "menuRepository": Represents an instance of MenuRepository that will provide CRUD operations for the Menu entity.
- "LOGGER": This is a static final logger instance from SLF4J (Simple Logging Facade for Java), used for logging and debugging.

The constructor of this class takes the first 2 arguments mentioned above. Their instances will be injected by Spring into our service class at the creation point with the use of the "@Autowired" annotation. For a better understanding of how some of the methods are created in this class we can take a look at the "authenticateUser" method.

As we can see in , the signature of the method indicates that this method is "public" and can be accessed by other classes and returns a Boolean value indicating that the authentication was a success or not. The next lines, after the method signature, are trying to fetch a user with the given username using the instance of the UserRepository and check if the user is existent or null. The next step is to check whether the password corresponds to the found user or not and returns the boolean value resulted by the line "return storedPassword.equals(password);".

The last part of this method is implemented to handle the case when the user is not found, or, better said, if the "Optional" does not contain an "User". For this case the method will return "false" and will log an error message indicating that no user was found with the help of the next line: "LOGGER.error("No user with username {} found",username);"



```

@Service
public class UserService {

    13 usages
    private final UserRepository userRepository;

    4 usages
    private final MenuRepository menuRepository;

    5 usages
    private static final Logger LOGGER = LoggerFactory.getLogger(UserService.class);

    @Autowired
    public UserService(UserRepository userRepository, MenuRepository menuRepository) {
        this.menuRepository = menuRepository;
        this.userRepository = userRepository;
    }

    1 usage
    public boolean authenticateUser(String username, String password) {
        Optional<User> userOptional = userRepository.findByUsername(username);
        if(userOptional.isPresent()){
            User user = userOptional.get();
            String storedPassword = user.getPassword();
            return storedPassword.equals(password);
        }
        else {
            LOGGER.error("No user with username {} found",username);
            return false;
        }
    }
}

```

Figure 5.21 UserService class.

- Controllers:

Transitioning to the "Controllers" package, we arrive at the gateway of our Spring Boot backend application. This package is in charge of defining and exposing API endpoints that allow external interactions with the functionality of our application. We construct a robust and accessible interface for clients to interact with our application by utilizing Spring's annotations such as "@RestController", "@RequestMapping", and "@CrossOrigin".

The major goal of the "Controllers" package is to serve as a link between the outside world and our application's internal business logic, which is included under the "Services" package. Each controller class acts as a conduit for certain API endpoints, bringing similar processes together under a logical URL structure.

The "@RestController" annotation turns our controller classes into RESTful web services that can receive HTTP requests and react with appropriate replies. The "@RequestMapping" annotation links particular URLs to controller methods and is frequently used with HTTP method annotations such as "@GetMapping", "@PostMapping", "@PutMapping", and "@DeleteMapping". The required service methods are then invoked by these methods to provide the desired functionality. Additionally, the "@CrossOrigin" annotation enables cross-origin resource sharing, allowing client apps from various domains

to safely use our API endpoints. This annotation helps to reduce potential security issues while ensuring that our backend integrates seamlessly with multiple front-end technologies.

We isolate the complexities of our application's business logic by using the "Controllers" package, making it available via a standardized and well-defined API. This architecture encourages modularity and follows the separation of concerns principles, with each controller class focusing entirely on the API's endpoint definitions and parameter validation.

Dependency injection facilitates the interaction between the "Controllers" and "Services" packages. Through the "@Autowired" annotation, each controller class obtains an instance of the relevant service class in its constructor. This allows controllers to use the business logic enclosed within the service methods, resulting in seamless integration.

### 5.2.3. React Frontend App

An interactive and user-friendly interface for engaging with the backend and the recommender system is provided by our React frontend application's dynamic interplay of components. Let's explore the key features of our frontend, which is essential for showcasing the results of various actions. The React frontend application is the user's entry point into the system's functions. It organizes the visual display of data, user inputs, and system reactions through a well-structured hierarchy of components. Its main objective is to give consumers a smooth and interesting experience while interacting with the recommender system and the backend application in real-time.

Consuming the RESTful API endpoints that the backend application has exposed is one of the frontend's primary functions. For data requests and transmissions between the frontend and backend, these endpoints act as gateways. The React frontend expertly makes API requests via asynchronous processes, obtaining data from the backend, and displaying it in a well-organized and user-friendly way. The frontend dynamically renders numerous components to show the results of various actions after getting data from the backend. These elements include menus, preferences, user profiles, and suggested menu items. Utilising React's component-based design, we gain modularity and reusability, which enables us to effectively manage and update various user interface components. Our frontend is additionally competent at processing user inputs and returning them to the backend for processing. The interface smoothly records user activities and sends them to the backend for additional processing, whether they are submitting preferences, creating new menus, or storing certain food items.

We'll look into how React components communicate, handle states, and ease data flow as we examine the frontend's conception and execution. The recommender system and frontend application work together to provide a seamless environment that allows users to engage with the system's features.

For a better understanding of the implementation process of each page, and the ability to navigate between these pages we must take a look at our core file "App.js". The file () represents the top-level structure of this web application that sets up the navigation. The component uses "Router" to set up the routing and "Header" and "Footer" to set up the header and footer for any page of the system.

Next we make use of the "Routes" component which contains multiple "Route" components, each specifying the path and the related React element to render when the path is accessed. Here are the paths and routes defined in our project:

- `"/`: This is the homepage which renders the "Login" component.
- `"/welcome"`: This route shows the "WelcomePage" component, when an user is successfully logged in.

- “/admin”: This route renders the “Admin” component, when an admin logs into the application to make changes to the users or nutritionists.
- “/formpage”: This shows the “FormPage” component. The first component created in order to collect the required data for our recommender system.
- “/viewpage”: Renders the “MenuList” component. This component renders all the menus corresponding to the user logged in.
- “/view-menu”: Renders the “ViewMenu” component, which displays the menu contents of the selected menu.
- “/view-menu-user”: Renders the “MenuUserView” component, which is similar to the previous one but specific to the user with the addition of an add button to properly select the desired items for his shopping list.
- “/welcome-nutritionist”: Renders the “WelcomeNtr” component, when a nutritionist is successfully logged in.
- “/user-menus”: Displays the “UserMenus” component, which fetches the menus related to one user in when a nutritionist wants to access them.
- “/viewntr”: Shows the “SearchNutritionists” component, page where users can search for nutritionists.
- “/register”: A registration page represented by the “Register” component.
- “/formmenu”: Renders the “WriteMenu” component, where nutritionist can prescribe menus by hand.

```
function App() {
  return (
    <div>
      <Router>
        <Header/>
        <div className="main-content">
          <Routes>
            <Route path="/" exact element = {<Login/>}/>
            <Route path="/welcome" element = {<WelcomePage/>}/>
            <Route path="/admin" element = {<Admin/>}/>
            <Route path="/formpage" element = {<FormPage/>}/>
            <Route path="/viewpage" element = {<MenuList/>}/>
            <Route path="/view-menu" element = {<ViewMenu/>}/>
            <Route path="/view-menu-user" element = {<MenuUserView/>}/>
            <Route path="/welcome-nutritionist" element = {<WelcomeNtr/>}/>
            <Route path="/user-menus" element = {<UserMenus/>}/>
            <Route path="/viewntr" element = {<SearchNutritionists/>}/>
            <Route path="/register" element = {<Register/>}/>
            <Route path="/formmenu" element = {<WriteMenu/>}/>
          </Routes>
        </div>
        <Footer/>
      </Router>
    </div>
  );
}

export default App;
```

Figure 5.22 App.js

Let's delve deeper into the frontend's implementation, by examining some of our key components of the application.

One of the most important components is the "FormPage". This component sets a form where users may input or modify their personal data. The data is initially updated in one backend service once the form is submitted, and then it is sent to the recommender system service, which offers a menu and certain metrics according to the user's preferences. The primary events that make the information gathering and transfer to the backend and recommender system are the "handleInputChange" and "handleSubmit" ().

The first function is called whenever a change of any input in the form has occurred and updates the state with the new value. The second function is called when the form is submitted. This function is an asynchronous function and makes requests, with the help of the "axios" methods, to our backend and recommender system in order to update the user and then if the update was successful to send the user object to the recommender system. If an error has occurred, the method will log them in the browser console. We also use a "loading" circle component to show our users that the recommender process is still in progress.

```
const [formData, setFormData] = useState(initialFormData);
const handleInputChange = (event) =>{
  const { name, value} = event.target;
  setFormData({...formData, [name]: value});
}
const handleSubmit = async (event) => {
  event.preventDefault();
  try{
    const response = await axios.put('http://localhost:8080/api/user/' + location.state.user.id,formData);
    if(response.status === 200){
      console.log('User updated',JSON.stringify(response.data));
      setLoading(true);
      const userObject = response.data;
      const sendResponse = await axios.post('http://localhost:7777/receive_user',userObject);
      if(sendResponse.status ===200){
        console.log('User sent successfully');
        setMenuItems(sendResponse.data.menu);
        setAccuracy(sendResponse.data.accuracy);
        setPrecision(sendResponse.data.precision);
        setRecall(sendResponse.data.recall);
        setF1(sendResponse.data.f1score);
        console.log('Accuracy', response.data.accuracy)
        setMenu(true);
      }else{
        console.log('Failed o send user object');
      }
    }else{
      console.log('Failed to update user');
    }
  }catch (error){
    console.log('An error occurred during user update:', error);
  } finally{
    setLoading(false);
  }
}
```

Figure 5.23 Handle user data and submit.

If the process has proven successful, the frontend application will render, with the help of the "MenuPage" component, the contents of the generated menu along with a feature that will let the user to save the menu if he wants to. This component represents a modal popup window that is displayed on top of the current page and displays the resulted meal plan according to the aforementioned template, by using the "Modal" component of the "react-bootstrap" library. In we will see the implementation of the component focusing on the HTML part of the "MenuPage.js" file. The meal plan contains 9 foods, each 3 of them being

classified as breakfast, lunch and dinner to properly show the template of our menu and each food will be displayed with their description and their serving size.

```
return (
  <Modal show={show} onHide={onClose}>
    <Modal.Header closeButton>
      <Modal.Title>Menu</Modal.Title>
    </Modal.Header>
    <Modal.Body>
      <div className="container">
        <h3>Breakfast</h3>
        <ul className="list-group">
          {breakfast.map(item => (
            <li className="list-group-item d-flex justify-content-between" key={item.ID}>
              <span>{item.Description[Object.keys(item.Description)[0]]}</span>
              <span>{item.ServingSize[Object.keys(item.ServingSize)[0]]} grams</span>
            </li>
          ))}
        </ul>
        <h3>Lunch</h3>
        <ul className="list-group">
          {lunch.map(item => (
            <li className="list-group-item d-flex justify-content-between" key={item.ID}>
              <span>{item.Description[Object.keys(item.Description)[0]]}</span>
              <span>{item.ServingSize[Object.keys(item.ServingSize)[0]]} grams</span>
            </li>
          ))}
        </ul>
        <h3>Dinner</h3>
        <ul className="list-group">
          {dinner.map(item => (
            <li className="list-group-item d-flex justify-content-between" key={item.ID}>
              <span>{item.Description[Object.keys(item.Description)[0]]}</span>
              <span>{item.ServingSize[Object.keys(item.ServingSize)[0]]} grams</span>
            </li>
          ))}
        </ul>
      </div>
    </Modal.Body>
    <Modal.Footer>
      <Button variant="secondary" onClick={onClose}>
        Close
      </Button>
      <Button variant="primary" onClick={handleSave}>
        Save
      </Button>
    </Modal.Footer>
  </Modal>
);
```

Figure 5.24 Menu popup component.

Together with the other components, these elements give a user-friendly experience to our users and nutritionists, each of them being able to perform certain tasks, like the menu generation, which was broken down above.

## Chapter 6. Testing and Validation

This chapter highlights the meticulous steps we've taken to assure the functionality, correctness, and performance of our application, but also marks a crucial turning point in our quest to develop a strong and dependable system. We explore the complicated environment of our project, verifying every aspect of it using a mix of thorough testing procedures and validation approaches.

Our testing and validation process covers a wide range of fields, each of which improves the system's overall quality. We start by looking at how the K-Nearest Neighbors (KNN) method, one of the fundamentals of our application's intelligence, is implemented. This method is one of the key components for incorporating user preferences in our meal generating system, by predicting the next appropriate foods that will satisfy our user.

Here, we discuss the specifics of the KNN evaluation metrics, such as accuracy, precision, recall, and F1 score. But before seeing the actual results of these metrics, we need to take a look at their implementation in the context of our recommender system ().

```
def calculate_metrics(knn,X_test,y_test,preferences):
    true_pred = 0
    total_pred = len(y_test)
    true_pos = 0
    false_pos = 0
    false_neg = 0
    lower_pref = preferences.replace(',',' ')
    pref = lower_pref.upper()
    print(pref.split())
    for i in range(total_pred):
        pred_label = knn.predict([X_test.iloc[i]])[0]
        if common_words(pred_label,y_test[i]):
            true_pred += 1
            if common_words(pred_label,pref):
                true_pos+=1
        else:
            if common_words(pred_label,pref):
                false_pos+=1
            else:
                false_neg+=1
    print(true_pos, false_pos,false_neg)
    accuracy = true_pred / total_pred
    acc = "{:.3f}".format(accuracy)
    precision = true_pos/(true_pos + false_pos + 1e-9)
    prec = "{:.3f}".format(precision)
    recall = true_pos/(true_pos + false_neg + 1e-9)
    rec = "{:.3f}".format(recall)
    f1_score = 2*(precision * recall)/(precision + recall + 1e-9)
    f1 = "{:.3f}".format(f1_score)
    return acc,prec,rec,f1
```

*Figure 6.1 Metrics computation.*

The method computes our desired metrics for evaluating our KNN model, by using the formulas presented in the previous chapters. Because our KNN model helps in the development of our recommender system, we concluded that for higher metrics results we need to

use a custom matching criterion, which is based on the presents of common words between our predicted labels and true labels. Another key element in the computation of these metrics was avoiding division with 0. That's why we chose to add for each denominator in the formula the small value of  $1e-9$ . After generating a menu, the metrics are computed and sent to the frontend along with the user's menu in case we want to print the results, which are the following:

- Accuracy = 0.328
- Precision = 0.750
- Recall = 0.005
- F1 score = 0.010

Another important aspect that I want to cover in this chapter is the API endpoints evaluation. This step becomes crucial in the testing and validation process, especially when it comes to web-based applications that communicate with backend servers. In this stage, Postman, a well-known API testing tool, is essential since it enables developers to evaluate the usability, precision, and resilience of the API endpoints in their application. A variety of situations and use cases may be validated by using Postman to examine API endpoints. We can evaluate various inputs, simulate user interactions, and watch how the software behaves. This iterative procedure helped us identify specific flaws, including improper data handling, unexpected behavior, or inaccurate replies.

An example of the usefulness of the Postman tool is illustrated in the , where we selected the method, we want to examine, as well as the URL of the API endpoint: “http://localhost:8080/api/user/1/shoppingList/items”. As the URL suggests, the method should return all the items that the user with ID 1 has in his shopping list, the real result being shown in the response section of the figure. We can also change the method and try using different URL, for example we can verify if the deletion of one menu is successful.

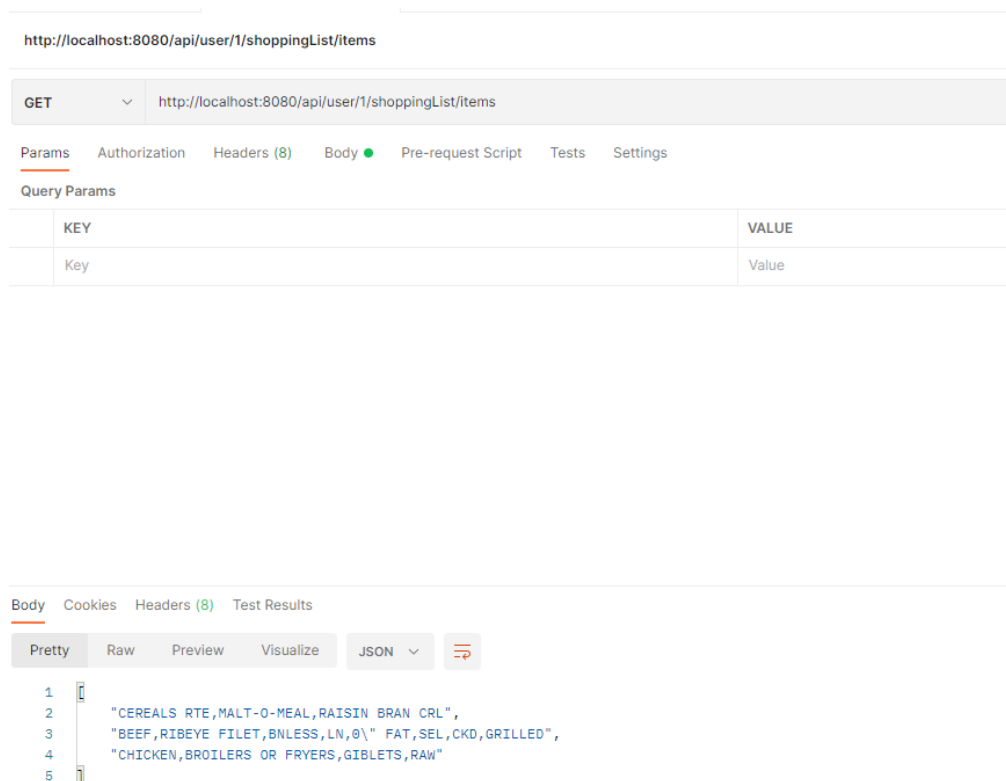


Figure 6.2 Postman result regarding shopping list items.

The last part I want to cover in this chapter, which also completes the whole testing and validation process, revolves around manual testing. While automated testing technologies give insightful information about code-level functionality, manual testing from the viewpoint of the user offers a distinctive viewpoint that supports the overall testing approach.

Validating user interactions is one of the most important components of manual testing. We navigate through the user interface of the system, take actions, and engage with various elements, features, and functionalities. This practical technique demonstrates the usability, responsiveness, and intuitiveness of the user interface components. It assists in locating problems like broken links, navigational challenges, or inconsistent visual design.

This validation process had a big impact on implementing and maintaining a functional system, which was mostly used while developing the frontend. While making small changes during the implementation and keeping our frontend application running, we saw the impact of every saved modification. One use case that stands out during the verification process is the update of the shopping list that corresponds to one user.

This use case revolves around the presence of the shopping list at the corner of the screen and how every addition or deletion of an item can be seen in real time in the list. We implemented the shopping list this way in order to give the users the availability of the shopping list no matter what page did we navigate to. This functionality, along with other user interactions will be presented in the next chapter.



## Chapter 7. User's Manual

This chapter contains step-by-step instructions for ensuring smooth installation and effective execution of the application. Whether you're a new user/nutritionist trying to get started or an experienced user/nutritionist looking for a refresher, this guide is designed to help you engage with the application's features and capabilities with ease. Our software is intended to improve the user's experience by simplifying complicated processes, giving useful insights, and speeding interactions. This guidebook will help you make the most of these features, from installation to utilization. We realize the importance of a user-friendly experience, therefore our user manual is filled with clear instructions, visual aids, and helpful hints to ensure your journey is simple and enjoyable.

### 7.1. Installation process

The following steps describe a complete installation method that involves setting up the required tools and components to ensure successful installation and effective use of our application. We will break down the installation process of each needed component of our system, by thorough examination of each step. Before delving into the setup of our system we need to ensure that our user has acquired the necessary tools in order to run our system. The system was implemented to run locally on every user's computer and will come in a “.zip” type of file, that will contain not only our backend and frontend, but also our recommender system component and the aforementioned dataset.

#### 7.1.1. Backend and Database connection setup

First, we need to have access to an integrated development environment such as IntelliJ IDEA or Eclipse (the next steps are illustrating the setup in the case of IntelliJ IDEA IDE). You also have to make sure that you have the appropriate JDK version installed, which should be higher or at least JDK 11. Another important resource that you should have on your computer is the MySQL Workbench or some other MySQL management tool. You also need to create a blank schema which then will be used by the backend application to operate based on the given name. To complete the connection between our backend and database we also need to install Maven which will manage any dependencies that the project has, with the help of the IDE. The last step revolves around changing the fields of the “application.properties” file (see instructions in *Figure 7.1 “application.properties” setup.*) to safely match your database password and username and also to choose the appropriate port and database name that you're using. After that hit the run button from your IDE and start your application.

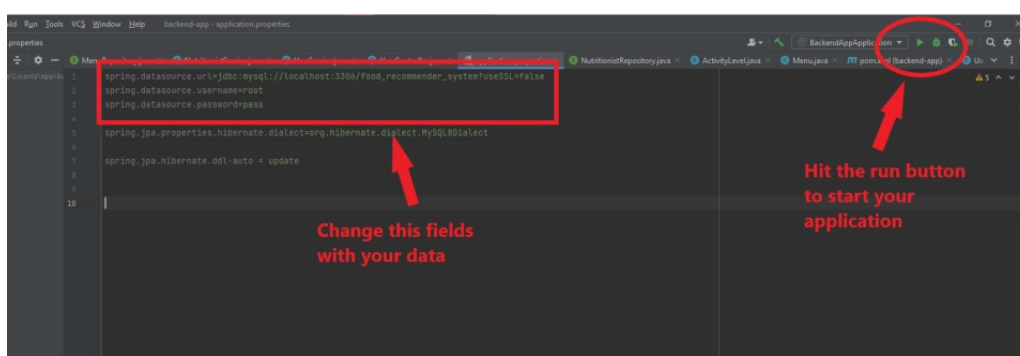


Figure 7.1 “application.properties” setup.

### 7.1.2. Frontend setup

The easiest step in our installation process is the frontend setup. I recommend you to use as an integrated development environment (IDE) the Visual Studio Code application, not only because is very intuitive and user-friendly but also because it will be a useful tool for installing and running our recommender system component. You should also have already installed the “Node.js”, which is another important requirement of this setup stage for running JavaScript outside the browser. Lastly, after completing these steps you should be able to start your frontend system by typing in the command “npm run” in the terminal of your IDE as shows.

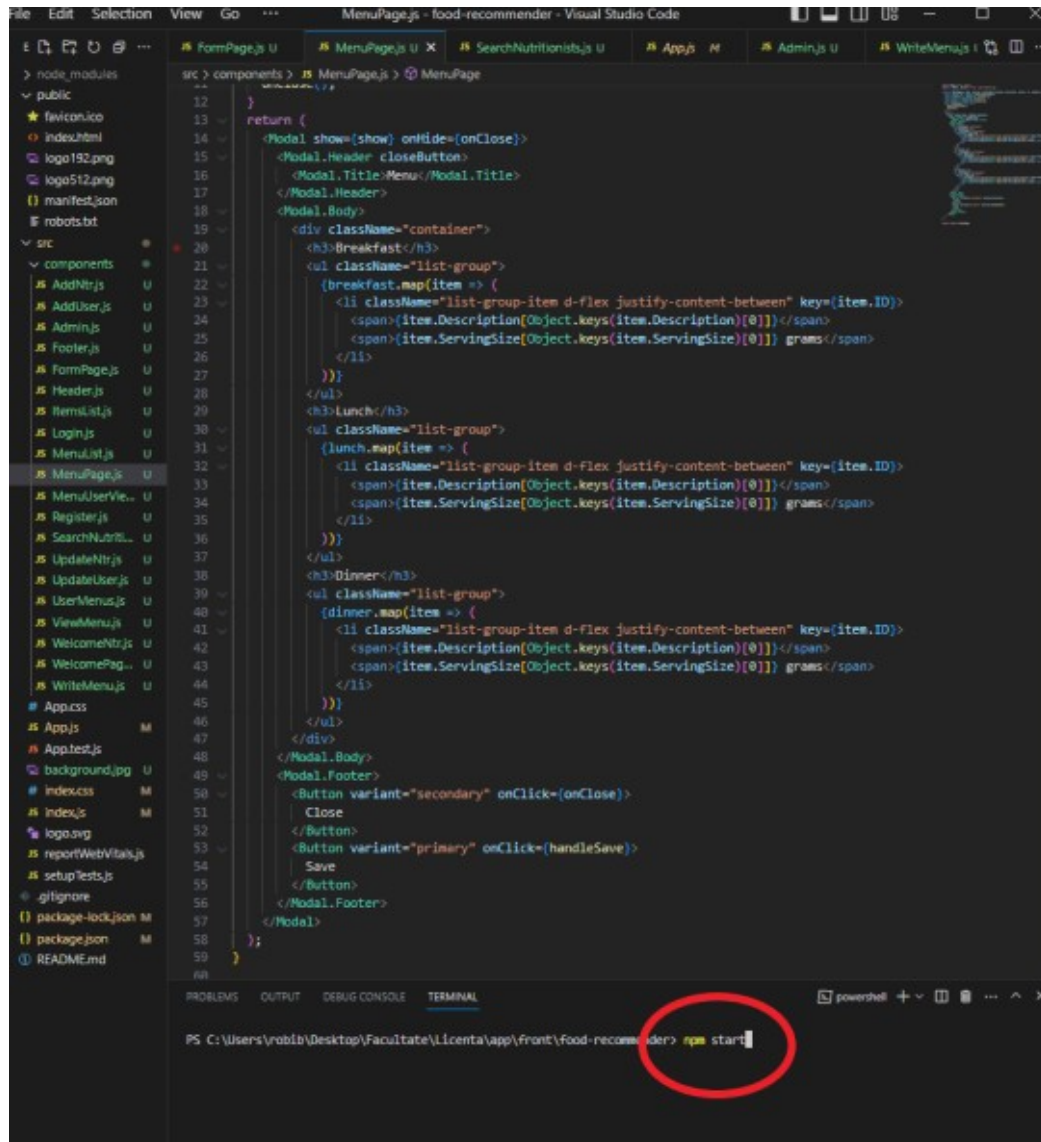


Figure 7.2 Starting the frontend.

### 7.1.3. Recommender system app

This is the last step in our setup and installation stage. Because this part of the system was implemented using Python, you are required to install a version of Python 3 along with the aforementioned libraries. This requires also the installation of an IDE, which as I previously mentioned in the section above, I recommend you to use again the Visual Studio

Code. Before installing all the required libraries that helped us in implementing this component, you need to have installed a virtual environment in order to isolate the dependencies for the recommender system. To start up the environment we make use of the command ".venv\Scripts\activate" illustrated in written in the console. The installation of all the libraries needed is done with the help of the command "pip", followed by the name or definition of the library. After that we can simply turn on our application with the "python app.py" command.

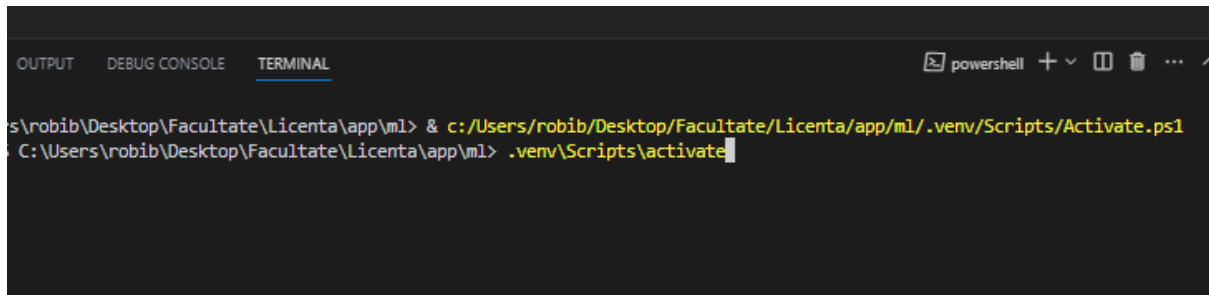


Figure 7.3 Virtual environment activation.

## 7.2. How to use our system

After setting up the system, we are ready to explore the use of the Food Recommender System. This section will focus on how the users can navigate our web application but also perform different operation based on their desires. The application starts after the "npm run" command was written in the console of our React component and it will open up the browser and render our Login Page *Figure 7.4 Login Page.*

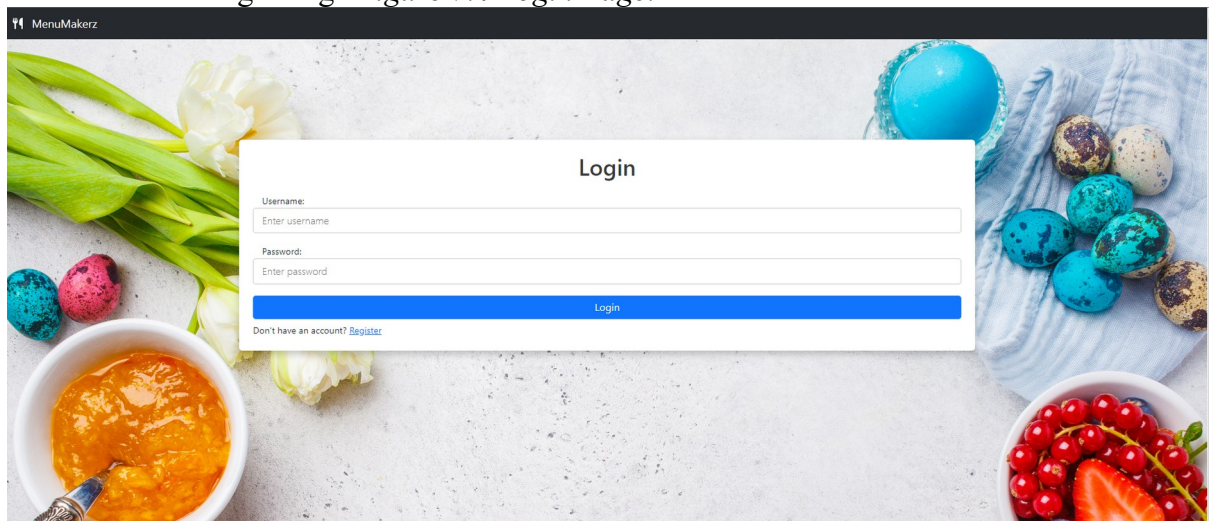


Figure 7.4 Login Page.

### 7.2.1. Login and Register

At this step our user must decide to either login if they have already created an account or create a new one through navigating to the Register Page by pressing the highlighted text under the Login form "Register". After navigating to this page, the user is required to input and submit their personal data in order to create a new account and enjoy the features of our application (*Figure 7.5 Register Page*). If a nutritionist wants to log in they need to have an already created account, they can register only if they contact our administrator. This feature allows us to block the access of fake nutritionists that don't have the required qualification for that position.



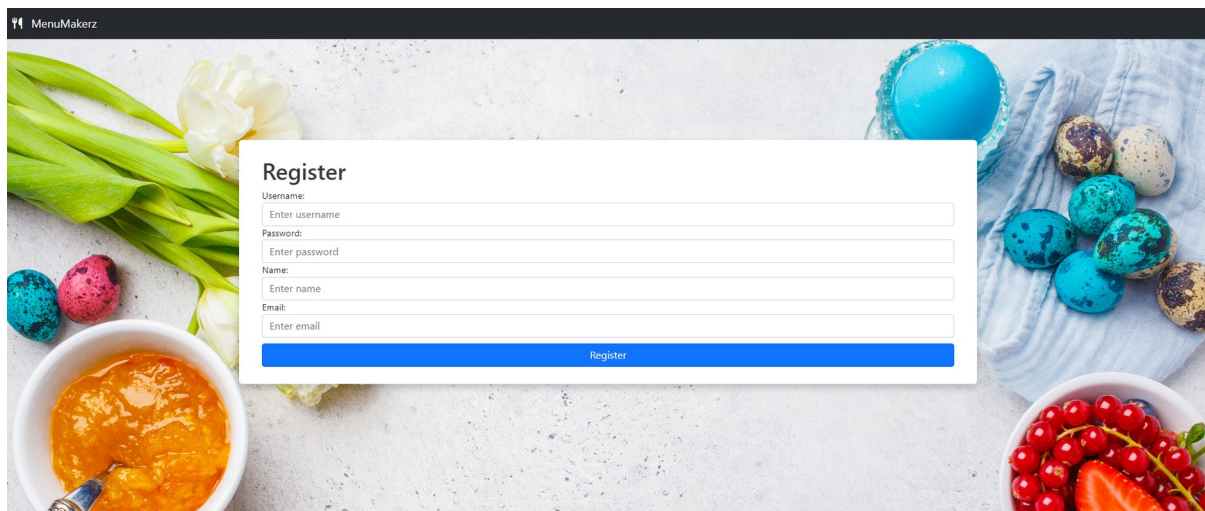


Figure 7.5 Register Page.

### 7.2.2. Client's manual

Here we will cover all the features that are available to our user, if he is a client. First the user will navigate to the Welcome Page () after a succesfull login and has the ability to chose his next option.

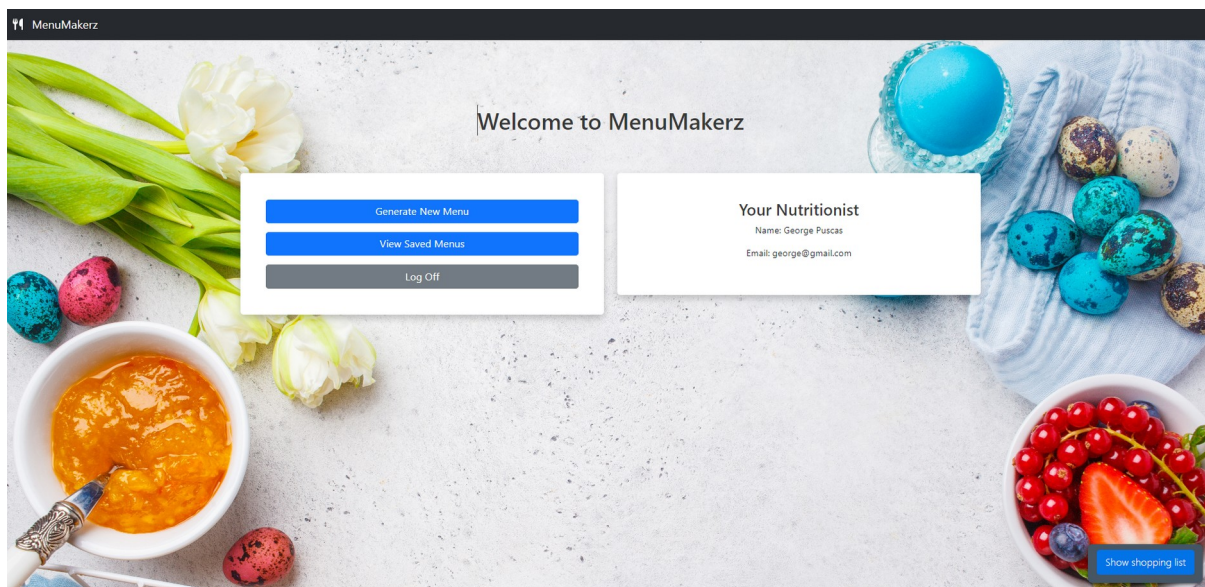


Figure 7.6 Welcome Page.

If the client is assign to a nutritionist then the nutritionist will be shown at the right part of the menu, if not the page will render a button that will let our user to navigate to the list of all nutritionists enlisted to our application and can choose either to contact them throught the associated email or not.

The client has the ability to choose either to generate a new menu using our recommender system, by pressing the "Generating New Menu" button. This will navigate our user to the Form Page which he has to complete in order to generate the meal plan. The illustrates our Form Page along with the generated menu resulted after the client submitted his personal details, which include the age, height, weight, personal preferences and many more. If the user is satisfied with the menu components he can save the meal plan.

**Menu**

**Breakfast**

MILK,LOWFAT,FLUID,1% MILKFAT,PROT FORT,W/	240 grams
ADDED VIT A & D	
CEREALS RTE,KELLOGG,KELLOGG'S SPL K	40 grams
BLUEBERRY	
PILIMS,RAW	65 grams

**Lunch**

CHICKEN BREAST TENDERS.UNCKD	100 grams
RICE,WHITE,MEDIUM-GRAIN,RAW,ENR	168 grams
MUSHROOMS,MOREL,RAW	66 grams

**Dinner**

BEEF,RND,TOP RND,STEAK,LN,1/8"	85 grams
FAT,SEL,CKD,BRLD	
ARCHWAY HOME STYLE COOKIES,FROSTY	28 grams
LEMON	
CRABAPPLES,RAW	55 grams

Close Save

Create Menu

Back to previous page

Figure 7.7 Form Page and generated meal plan.

For accessing the already generated menus or the prescribed menus that were created by the assigned nutritionist, if that's the case, the client can access his list of menus by pressing the "View Saved Menus" button, which will send our user to the Menus Page. As the shows, the user can delete the saved menus or examine the contents of each menu.

**Menus**

Menu 1

View Menu Delete

Menu 2

View Menu Delete

Menu 6

View Menu Delete

Menu 17

View Menu Delete

Menu 18

View Menu Delete

Menu 19

View Menu Delete

Hide shopping list

CEREALS RTE,MALT-O-MEAL,RAISIN BRAN CRL	X
BEEF,RIBEYE FILET,BNLESS,LIN,0" FAT,SEL,CKD,GRILLED	X
CHICKEN,BROILERS OR FRYERS,GIBLETS,RAW	X

Figure 7.8 Menus Page.

As we can observe at the bottom of the screen, the client has access to his Shopping List which can be updated by adding foods from the View-Menu Page () or delete them from the list using the "X" button. He can also choose to show or hide the shopping box.



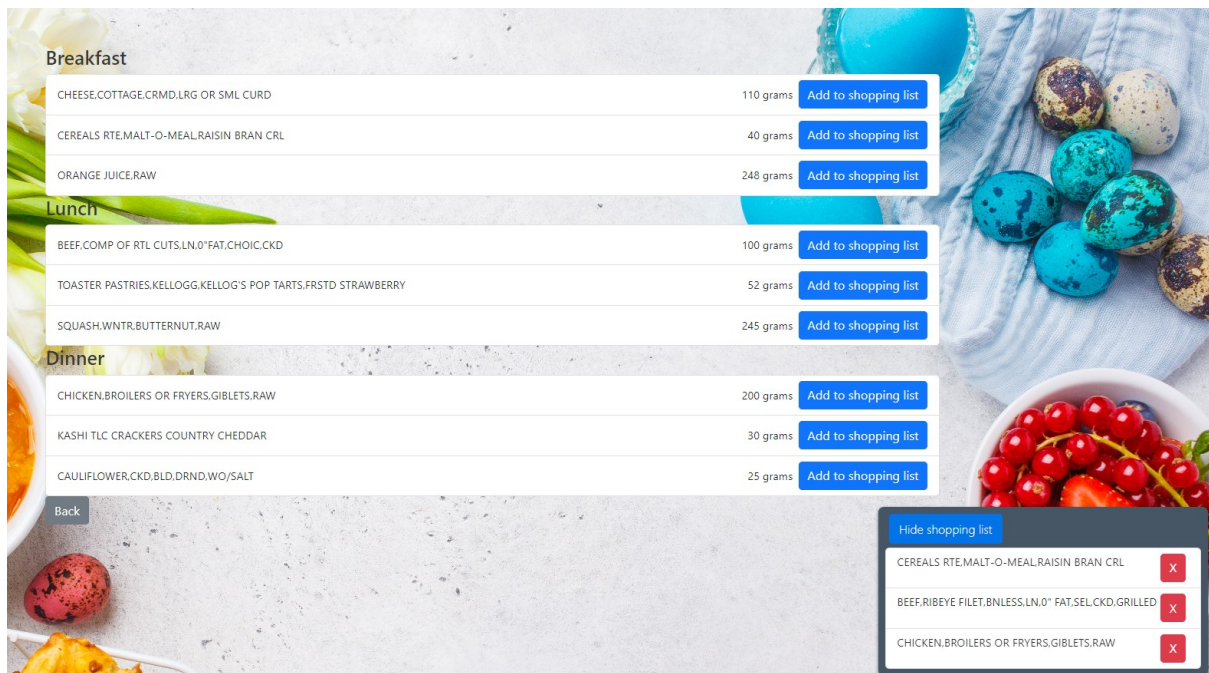


Figure 7.9 View-Menu Page.

### 7.2.3. Nutritionist's manual

This section will present all the feature available to the nutritionist after a succesfull login. Similar to the clients, if the login was succesfull the nutritionist will be sent to the Welcome Page () and will be shown all the clients assigned to him, along with their information. If there are no clients, then a message will be shown.

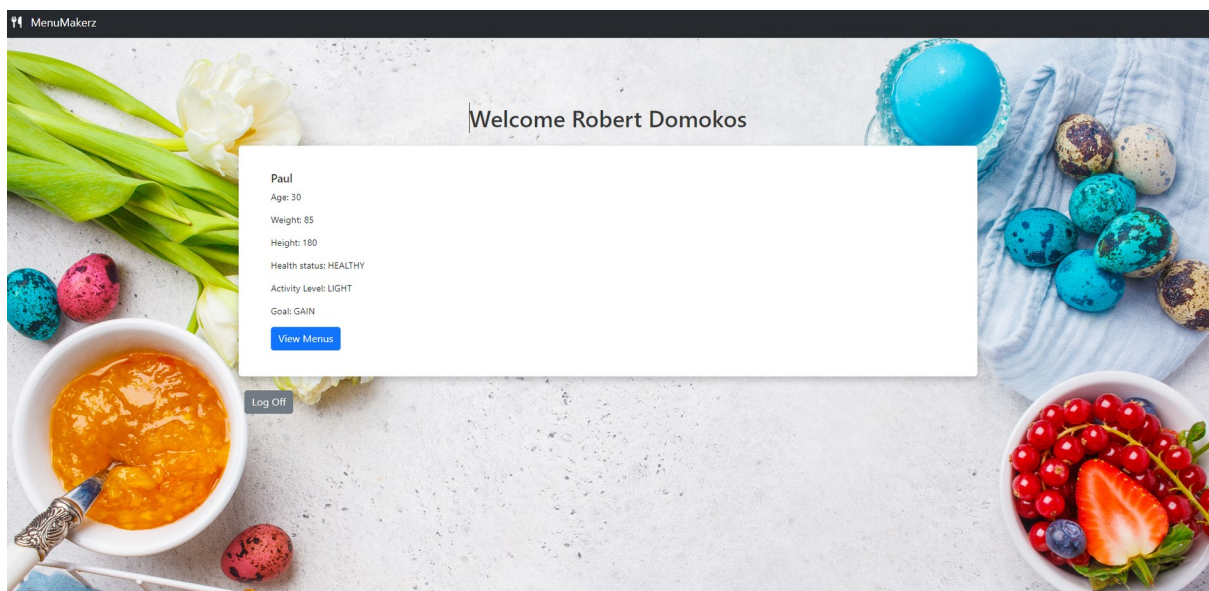


Figure 7.10 Welcome Page (Nutritionist).

The nutritionist can access the client's menus by pressing the "View Menus" button corresponding to each client, then they will navigate to the View-Client Page. As the shows, the nutritionist can access the menus components or delete them if they are not in the favor of their clients. They can also generate a new menu using our recommender system in a similar fashion as presented in the client's manual. If the nutritionist wantss to prescribe a menu by

hand they can do that by pressing the "Prescribe New Menu" and navigate to the Prescribe-Menu Page. As the suggest, the nutritionists have to complete the form in order to add the menu to the client's menus list.

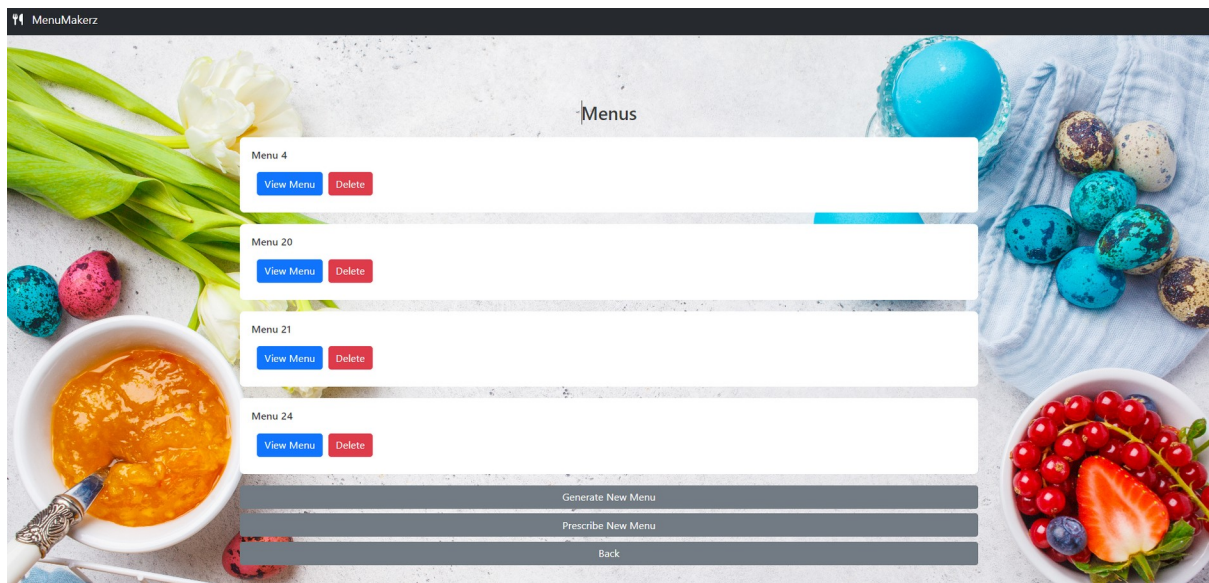


Figure 7.11 View-Client Page.

The screenshot shows the "Write Menu Form" page. The background features a collage of food items including tulips, blue eggs, and a bowl of blueberries. The form is organized into three sections: "Breakfast", "Lunch", and "Dinner". Each section contains three rows of input fields for "Description" and "Serving Size". At the bottom of the form, there are two buttons: "Add Menu" (blue) and "Back to previous page" (grey).

Figure 7.12 Prescribe Menu Page.

## Chapter 8. Conclusions

In conclusion, the creation and application of the Food Recommender System have been significant efforts with the goal of offering users customized and wholesome meal plans catered to their preferences and dietary requirements. In order to produce a seamless and user-friendly experience, we have concentrated on integrating a variety of technologies, from frontend React interfaces to backend Spring Boot applications. This path has not only produced a useful application but has also given us insightful knowledge in the fields of nutrition, web development, and machine learning.

Our main contribution involves the development of a comprehensive food recommendation platform that connects users, backend services, and the recommender system. We have successfully constructed a system that creates personalized menus, gives nutritional insights, and assists users in making educated food choices by using the capabilities of technologies such as Spring Boot, React, and Python libraries. The integration of K-Nearest Neighbors (KNN) and Analytical Hierarchy Process (AHP) approaches in our recommender system adds sophistication to the application, allowing it to adapt to the demands of individual users.

Our Food Recommender System produced encouraging results, proving its capacity to develop meals that correspond to users' dietary choices and nutritional requirements. Accuracy, precision, recall, and F1-score, among other measures used to evaluate the performance of the KNN method, haven't showed desirable results, but have revealed insights into the practicality of our recommendation model. However, it is critical to recognize that the quality and quantity of input data, as well as the complexity of human preferences, may all have an impact on the accuracy of such systems. Further data collecting and model optimization improvements might increase the accuracy and performance of the recommendations.

The Food Recommender System, like any software project, has room for future additions and enhancements that not only will improve our results but will also provide a faster response from the system. While the system's present implementation combines K-Nearest Neighbors and Analytical Hierarchy Process (AHP) approaches, experimenting with more sophisticated machine learning techniques may result in increased recommendation accuracy. Deep learning, collaborative filtering, and hybrid models, for example, might aid in capturing the complicated links between user preferences and food qualities. These sophisticated techniques have the potential to provide accurate and context-aware suggestions, hence improving overall user experience.

User feedback is critical in developing and optimizing any recommendation system. By incorporating a feedback mechanism into the Food Recommender System, users will be able to score and evaluate recommended meals as well as individual food items. This feedback loop enables the system to learn from user choices and provide appropriate recommendations. By integrating user feedback on a constant basis, the system may develop its models and give increasingly precise and personalized meal recommendations.

Another improvement that could become crucial for future versions of our Food Recommender System is the integration of external data sources. By utilizing external data sources, the system's database may be enriched with accurate and up-to-date nutritional information. Integrating data from credible food databases, nutritional APIs, or health-related platforms can provide the system with complete and dependable information about diverse



food items. This connection can improve the precision of nutritional calculations and allow the system to give users more exact dietary information.

The next update of our system might focus on providing consumers with more control and customization or personalization possibilities. Allowing users to identify precise dietary limitations, allergies, cultural preferences, and even providing a certain time for a meal to be consumed according to their daily plans might result in suggestions that are highly personalized. In addition, the system can adjust to changes in user preferences over time, resulting in a dynamic and developing meal suggestion experience.

As technology advances, our Food Recommender System demonstrates the power of combining data, algorithms, and user-centric design to improve the way we approach our eating habits. We are set to uncover even larger opportunities in encouraging healthier lifestyles through personalized nutrition advice with ongoing refinement and innovation.

## **Bibliography**

- [1] J. M. Ordovas, L. R. Ferguson, E. S. Tai, and J. C. Mathers, "Personalised nutrition and health," *Brit. Med. J.*, vol. 361, Jun. 2018, Art. no. bmj.k2173.
- [2] Nuttall FQ. Body mass index: obesity, BMI, and health: a critical review. *NutrToday*. 2015;50:117–28.
- [3] <https://www.healthline.com/health/what-is-basal-metabolic-rate#How-many-calories-do-you-need-every-day?>
- [4] T. Cioara, I. Anghel, I. Salomie, L. Barakat, S. Miles, D. Reidlinger, A. Taweel, C. Dobre, and F. Pop, "Expert system for nutrition care process of older adults," *Future Gener. Comput. Syst.*, vol. 80, pp. 368-383, Mar. 2016.
- [5] F. Rehman, O. Khalid, N. ul Haq, A. ur Rehman Khan, K. Bilal, and S. A. Madani, "Diet-right: A smart food recommendation system," *KSII Trans. Internet Inf. Syst.*, vol. 11, no. 6, pp. 2910-2925, 2017.
- [6] Haytowitz, David B.; Ahuja, Jaspreet K.C.; Wu, Xianli; Somanchi, Meena; Nickle, Melissa; Nguyen, Quyen A.; Roseland, Janet M.; Williams, Juhi R.; Patterson, Kristine Y.; Li, Ying; Pehrsson, Pamela R. (2019). USDA National Nutrient Database for Standard Reference, Legacy Release. Nutrient Data Laboratory, Beltsville Human Nutrition Research Center, ARS, USDA. <https://doi.org/10.15482/USDA.ADC/1529216>. Accessed 2023-08-16.
- [7] F. Ricci, L. Rokach, and B. Shapira, "Recommender systems handbook," in *Recommender Systems Handbook*, F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, Eds. New York, NY, USA: Springer, 2011, ch. 1, pp. 1-35.
- [8] A. Gunawardana and G. Shani, "A survey of accuracy evaluation metrics of recommendation tasks," *J. Mach. Learn. Res.*, vol. 10, pp. 2935-2962, Dec. 2009.
- [9] Toledo RY, Alzahrani AA, Martinez L (2019) A food recommender system considering nutritional information and user preferences. *IEEE Access* 7:96695–96711
- [10] A. Ishizaka and S. Siraj, "Are multi-criteria decision-making tools useful? An experimental comparative study of three methods," *Eur. J. Oper. Res.*, vol. 264, no. 2, pp. 462-471, 2018.
- [11] C. L. Hwang and K. Yoon, *Multiple Attribute Decision Making: Methods and Applications*. New York, NY, USA: Springer-Verlag, 1981.

- [12] [https://www.researchgate.net/publication/373015635\\_What\\_is\\_Machine\\_Learning](https://www.researchgate.net/publication/373015635_What_is_Machine_Learning)
- [13] Toh, C.; Brody, J.P. Applications of Machine Learning in Healthcare. 2021.
- [14] D. Kirk, E. Kok, M. Tufano, B. Tekinerdogan, E. J. M. Feskens, and G. Camps, "Machine Learning in Nutrition Research," in *Advances in Nutrition*, vol. 13, no. 6, pp. 2573–2589, Nov. 2022.
- [15] Saaty, T.L. The Analytic Hierarchy Process—What it is and how it is used. *Math. Model.* 1987,9, 161–176.
- [16] [https://www.researchgate.net/publication/228686398\\_k-Nearest\\_neighbour\\_classifiers](https://www.researchgate.net/publication/228686398_k-Nearest_neighbour_classifiers)
- [17] Peterson, L. E. K-nearest neighbor. *Scholarpedia* 4, 1883 (2009).
- [18] <https://www.analyticsvidhya.com/blog/2021/07/metrics-to-evaluate-your-classification-model-to-take-the-right-decisions/>
- [19] G. Agapito, M. Simeoni, B. Calabrese, I. Caré, T. Lamprinoudi, P. H. Guzzi, A. Pujia, G. Fuiano, and M. Cannataro, "DIETOS: A dietary recommender system for chronic diseases monitoring and management," *Comput. Methods Programs Biomed.*, vol. 153, pp. 93104, Jan. 2018.
- [20] F. Mata, M. Torres-Ruiz, R. Zagal, G. Guzman, M. Moreno-Ibarra, and R. Quintero, "A cross-domain framework for designing healthcare mobile applications mining social networks to generate recommendations of training and nutrition planning," *Telematics Informat.*, vol. 35, no. 4, pp. 837853, 2018.
- [21] D. Ribeiro, J. Machado, M. J. M. Vasconcelos, E. Vieira, and A. C. de Barros, "Souschef: Mobile meal recommender system for older adults," in *Proc. 3rd Int. Conf. Inf. Commun. Technol. Ageing Well E- Health*, 2017, pp. 36-45.
- [22] <https://www.fitbit.com/global/eu/home>
- [23] <https://www.freecodecamp.org/news/grabs-front-end-guide-for-large-teams-484d4033cc41>
- [24] <https://www.freecodecamp.org/news/mindset-lessons-from-a-year-with-react-1de862421981>

[25] <https://www.simplilearn.com/top-python-libraries-for-data-science-article>

[26] <https://www.jmlr.org/papers/volume12/pedregosa11a/pedregosa11a.pdf>