INSTITUTO POLITÉCNICO DE BRAGANÇA
Escola Superior de Tecnologia e Gestão

# Implementation of a multiplatform system for defining and monitoring of diets

**Paulo Pereira - a34513**

Project presented to the School of Technology and Management in the scope of the Master in Informatics.

Supervisors:
Prof. Maria João Varanda
Prof. Paulo Alves

Bragança
October 2023


# Implementation of a multiplatform system for defining and monitoring of diets

**Paulo Pereira - a34513**

Project presented to the School of Technology and Management in the scope of the Master in Informatics.

Supervisors:

Prof. Maria João Varanda

Prof. Paulo Alves

Bragança

October 2023

# Dedication

To my family,

My parents, who have always supported me and believed in my dreams. Maria, whose encouragement was fundamental to the successful completion of this journey.

To my friends, with a special mention to Sofia, for making this academic journey memorable.

With heartfelt thanks,

# Acknowledgment

This project is a collective effort, and I am deeply appreciative of everyone who played a role in its completion.

Particularly, my supervisor, Professor Maria João Varandas, who allowed me to develop this project. I am grateful for the guidance, support, patience, and accessibility that were invaluable throughout this journey.

To my co-supervisor, Professor Paulo Alves, I extend my appreciation for your contributions to this work and your important involvement.

I am also grateful to Escola Superior de Tecnologia e Gestão for providing the necessary resources and environment that allowed this project.

I would like to express my gratitude to Aquaevitae who initiated this project. Your request and collaboration provided a real-world context that enriched the project. The project AquaeVitae was supported by "FCT – Fundação para a Ciência e a Tecnologia" and "Fundação BPI La Caixa" (POCI-01-0145-FEDER-031309), under the title: "Aquae Vitae - Água Termal Como Fonte de Vida e Saúde". More informations about this project can be found at https://aquaevitae.pt/.

Additionally, I extend my heartfelt appreciation to the nutritionist, Dr. Filipe Ferreira, who shared invaluable scientific knowledge that contributed significantly to the success of this work.

A special thanks goes to my colleague, Henrique, who conducted crucial work for this project and engaged in productive discussions during our collaboration.

To my co-workers, thank you for accommodating my academic pursuits and office responsibilities. Your support allowed me to combine academia and work in a more

efficient way.

# Abstract

The AquaeVitae project innovatively promotes health and prevents illness, focusing on thermal water-based dietary recommendations. It features an intelligent system, split into a web platform for nutritionists and administrators, and a mobile app for patients, enhancing nutritional planning and client interaction.

It was decided to split the system into two parts: a web system for nutritionists and administrators, and a mobile system for patients. Nutrition apps are increasingly popular as they help clients manage their diet and make healthier food choices. Requirements were gathered for each user of the system and use case, entity-relationship and architecture diagrams were created. Prototypes were also developed for the web and mobile layout.

This project is centred on the Web application for the nutritionist, using technologies such as Vue.js, TypeScript and Node.js. Our project offers an intuitive layout for managing nutritional plans by simplifying the creation, editing, and updating of health plans. Moreover, we allow easy interaction with patient reports and food diaries. Last, this Web Application was designed to allow an easy update of the food database with new foods, particularly unique thermal-based foods.

**Keywords:** AquaeVitae Project, Thermal Water-Based Dietary Recommendations, Web Platform, Nutritional Planning, Vue.js, TypeScript, Patient Interaction and Management, Food Database (with Thermal-Based Foods)

# Resumo

O projeto AquaeVitae promove de forma inovadora a saúde e previne a doença, centrando-se nas recomendações dietéticas baseadas na água termal. Trata-se de um sistema inteligente, dividido numa plataforma web para nutricionistas e administradores e numa aplicação móvel para pacientes, que potencia o planeamento nutricional e a interação com o cliente.

Foi decidido dividir o sistema em duas partes: um sistema Web para nutricionistas e administradores e um sistema móvel para pacientes. As aplicações de nutrição são cada vez mais populares, uma vez que ajudam os clientes a gerir a sua dieta e a fazer escolhas alimentares mais saudáveis. Foram recolhidos requisitos para cada utilizador do sistema e foram criados diagramas de casos de uso, de entidade-relacionamento e de arquitetura. Foram também desenvolvidos protótipos para o layout Web e móvel.

Este projeto centra-se na aplicação Web para o nutricionista, utilizando tecnologias como Vue.js, TypeScript e Node.js. O nosso projeto oferece um layout intuitivo para a gestão dos planos nutricionais, simplificando a criação, a edição e a atualização dos planos de saúde. Além disso, permitimos uma fácil interação com os relatórios e diários alimentares dos pacientes. Por último, esta aplicação Web foi concebida para permitir uma fácil atualização da base de dados de alimentos com novos alimentos, em particular alimentos de base termal.

**Palavras-chave:** Projeto AquaeVitae, Recomendações Dietéticas Baseadas em Água Termal, Plataforma Web, Planeamento Nutricional, Vue.js, TypeScript, Interação e Gestão de Pacientes, Base de Dados de Alimentos (com Alimentos de Água Termal)

# Contents

**1 Introduction** **1**

**2 Context** **3**

2.1 Nutrition Apps: requirements . . . . . 3

2.1.1 Nutrition Apps: features . . . . . 3

2.1.2 Nutrition Apps: trustworthiness . . . . . 4

2.2 Nutrition Apps: examples . . . . . 5

2.2.1 Alimente-se . . . . . 5

2.2.2 MyFitnessPal . . . . . 8

2.2.3 Nutrients . . . . . 11

2.2.4 Nutrium . . . . . 11

2.2.5 Conclusions . . . . . 12

**3 Modeling and Technologies** **15**

3.1 Modeling . . . . . 15

3.1.1 Requirements . . . . . 15

3.1.2 Use Cases . . . . . 18

3.1.3 ER Diagram . . . . . 18

3.1.4 Architecture Diagram . . . . . 20

3.1.5 WEB Mockups . . . . . 20

3.1.6 Mobile Mockups . . . . . 24

3.1.7 Application Programming Interface . . . . . 26

3.2 Technologies . . . . . 28
3.2.1 Typescript . . . . . 28
3.2.2 Node.js . . . . . 29
3.2.3 Vue.js . . . . . 29
3.2.4 Axios . . . . . 31
3.2.5 Quasar Framework . . . . . 32
3.3 Conclusions . . . . . 32

**4 Implementation** **35**
4.1 Structure . . . . . 35
4.2 Router . . . . . 36
4.3 Services . . . . . 36
4.4 Types . . . . . 40
4.5 Views . . . . . 40
4.6 Components . . . . . 42
4.6.1 Base . . . . . 42
4.6.2 Table . . . . . 43
4.6.3 List . . . . . 46
4.6.4 Autocomplete . . . . . 47
4.6.5 Conclusion . . . . . 48

**5 Results** **49**
5.1 Developed Work . . . . . 49
5.2 Conclusions . . . . . 61

**6 Conclusions** **63**
6.1 Future Works . . . . . 63

**A Use Cases Diagram** **A1**

**B Base Component** **B1**

**C Table Component** **C1**

**D Autocomplete Component** **D1**

# List of Figures


2.1 *Alimente-se*: representative images of the application's layout. . . . . 7
2.2 *MyFitnessPal*: representative images of the application's layout. . . . . 10

3.1 List Example . . . . . 21
3.2 Patient List . . . . . 22
3.3 Nutritional Plan Page . . . . . 23
3.4 Form example . . . . . 23
3.5 Home . . . . . 24
3.6 Nutritional . . . . . 24
3.7 Diary . . . . . 24
3.8 Profile . . . . . 25
3.9 Results . . . . . 25
3.10 Form example . . . . . 25

4.1 Project Structure . . . . . 35
4.2 Header with Title . . . . . 42
4.3 Header Without Title . . . . . 42
4.4 Table Component . . . . . 46
4.5 List Component . . . . . 46
4.6 Autocomplete . . . . . 48

5.1 Appointment List . . . . . 50
5.2 Patient List . . . . . 51

5.3 Patient Menu . . . . . 52
5.4 Anthropometric Data . . . . . 52
5.5 Nutritional Plan . . . . . 53
5.6 Food List . . . . . 54
5.7 Meal List . . . . . 55
5.8 Meal Detail Dialog . . . . . 55
5.9 Managment Menu . . . . . 56
5.10 All Users List . . . . . 57
5.11 Type of Meals List . . . . . 57
5.12 Type of Meals Form . . . . . 58
5.13 Patient File Form Example 1 . . . . . 59
5.14 Patient File Form Example 2 . . . . . 60
5.15 Patient File Form Example 3 . . . . . 60
5.16 Form Validation . . . . . 61

# Chapter 1

# Introduction

The Aquae Vitae project [1], with its multidisciplinary approach, focuses on health promotion and disease prevention through various innovative lines of action. It explores the therapeutic effects of thermal water, develops functional foods and beverages incorporating natural mineral waters and bioactive compounds, and creates sustainable thermal water-based cosmetics. A key part of this project is the creation of an intelligent system for automatic individual suggestion of thermal-based healthy eating, aimed at enhancing the field of nutritional planning. [2]

Thermal-based foods integrate thermal waters (TW) from natural springs into their recipes, enriching them with minerals. An example is the adaptation of Portugal's traditional Biju bread using TW from Chaves. This inclusion enhances the bread's mineral content, like potassium and magnesium, without altering its fundamental nutritional composition. This innovative use of TW in food products like bread indicates potential for both nutritional enhancement and regional culinary innovation [3].

Our web application, is designed to revolutionize the way healthy eating is approached. It provides an intelligent system for automatically suggesting individual thermal-based healthy eating options. For nutritionists, it offers tools to generate detailed reports, facilitating the creation of personalized and effective meal plans. The system leverages artificial intelligence to cater to dietary restrictions and preferences, ensuring diverse and engaging meal options.

This document focuses on the development of the frontend for this web application, complementing the backend and mobile app components developed by other students. By integrating thermal-based foods, a key objective of the Aquae Vitae Project, our application aims to enhance the experience for both nutritionists and patients, aligning with the broader goals of the project.

# Chapter 2

# Context

Nutrition apps have become increasingly popular as customers recognize their value in achieving and maintaining better health through improved dietary habits. As people become more health-conscious, the demand for nutrition apps continues to grow, making them an essential part of the modern health and wellness landscape.

## 2.1 Nutrition Apps: requirements

Nutrition apps play a significant role in helping customers manage their diets and make healthier food choices. These apps offer a variety of features and benefits that customers find valuable, such as meal tracking, nutritional information, goal setting, customized meal plans, food diaries, nutrient analysis, progress tracking, recipe ideas, and social support.

### 2.1.1 Nutrition Apps: features

One key feature of nutrition apps is meal tracking, which allows users to monitor their daily food intake.

Additionally, nutrition apps provide detailed information about macronutrients (carbohydrates, proteins, and fats) and micronutrients (vitamins and minerals) improving the

ability of users to make informed choices about what they eat.

Customized meal plans are another valuable aspect of nutrition apps. This customization ensures that users can align their nutrition with their specific needs and objectives, whether they are focused on weight loss, muscle gain, or improved dietary habits. Moreover, they promote higher engagement with the nutritional plan.

Furthermore, nutrition apps often provide users with progress-tracking tools, allowing them to monitor their dietary and health-related achievements over time. These apps can be highly motivating, helping users stay on track with their nutrition goals and fostering a sense of accomplishment.

### 2.1.2 Nutrition Apps: trustworthiness

In addition to their practical features, the trustworthiness of a nutrition app is paramount. Users depend on these apps for accurate information and guidance regarding their health and diet. Several factors contribute to the trustworthiness of a nutrition app:

1. Credible Sources: A trustworthy nutrition app should source its information from reputable government health agencies, academic institutions, and registered dietitians or nutritionists.

2. Accurate Nutritional Information: The app should provide precise nutritional data for foods, sourced from reliable databases and updated regularly.

3. User Reviews and Ratings: Reading user reviews and ratings can offer insights into the app's reliability, although it's essential to consider a variety of opinions.

4. Privacy and Security: The app should have a strong privacy policy and robust security measures to protect user data.

5. Scientific References and Medical Disclaimer: Including scientific references and a clear medical disclaimer demonstrates the app's commitment to evidence-based information and acknowledges its limitations as a substitute for professional medical advice.

## 2.2 Nutrition Apps: examples

We have reviewed the existing market to guide the decision-making process for this project. Nutrition apps and software can vary widely in their focus and features. Most commonly, nutrient apps are more "patient-based" and were designed for individuals seeking to track their own nutrition and fitness goals. Less common are the applications primarily designed for nutrition professionals, such as dietitians and nutritionists, to help them manage their practices and work with clients effectively.

We have selected four examples to elucidate the state-of-the-art explored in sections 2.2.1, 2.2.2, 2.2.3 and 2.2.4. For each application, we have listed **features**, **layout** and **rating**.

### 2.2.1 Alimente-se

The app starts with a page for creating an account, where it requests information about the user goals for using the app (whether it's for weight loss or weight gain), target weight, and weekly weight loss goal (ranging from 250g to 1kg per week). After completing the registration, a new page appears, providing the user daily calorie consumption goal based on their inputs.

#### Features

*Alimente-se* has free and premium features listed below.

Free:

- Daily food diary with a list of all consumed foods, calculating consumed calories.

Premium:

- Set goals for carbohydrates, proteins, and fats.
- Food analysis, including a scanner that uses the phone's camera to suggest ingredients to add to the diary.

- Recipes for daily meals.

### Layout

The app's layout includes a menu at the bottom of the screen, with all pages presented in a list format. When the diary is opened, the current day is displayed, and the user can navigate to previous days. On this same page, the user can add ingredients to various meals throughout the day (Breakfast, Lunch, Dinner, Snacks). Additionally, it is possible to log exercises performed and track water consumption. In Figure 2.1, we present representative images of the application's layout.

### Ratings

- Android: 4.5 out of 5 based on 2.5 million ratings.
- iOS: 4.7 out of 5 based on 1.3 million ratings.

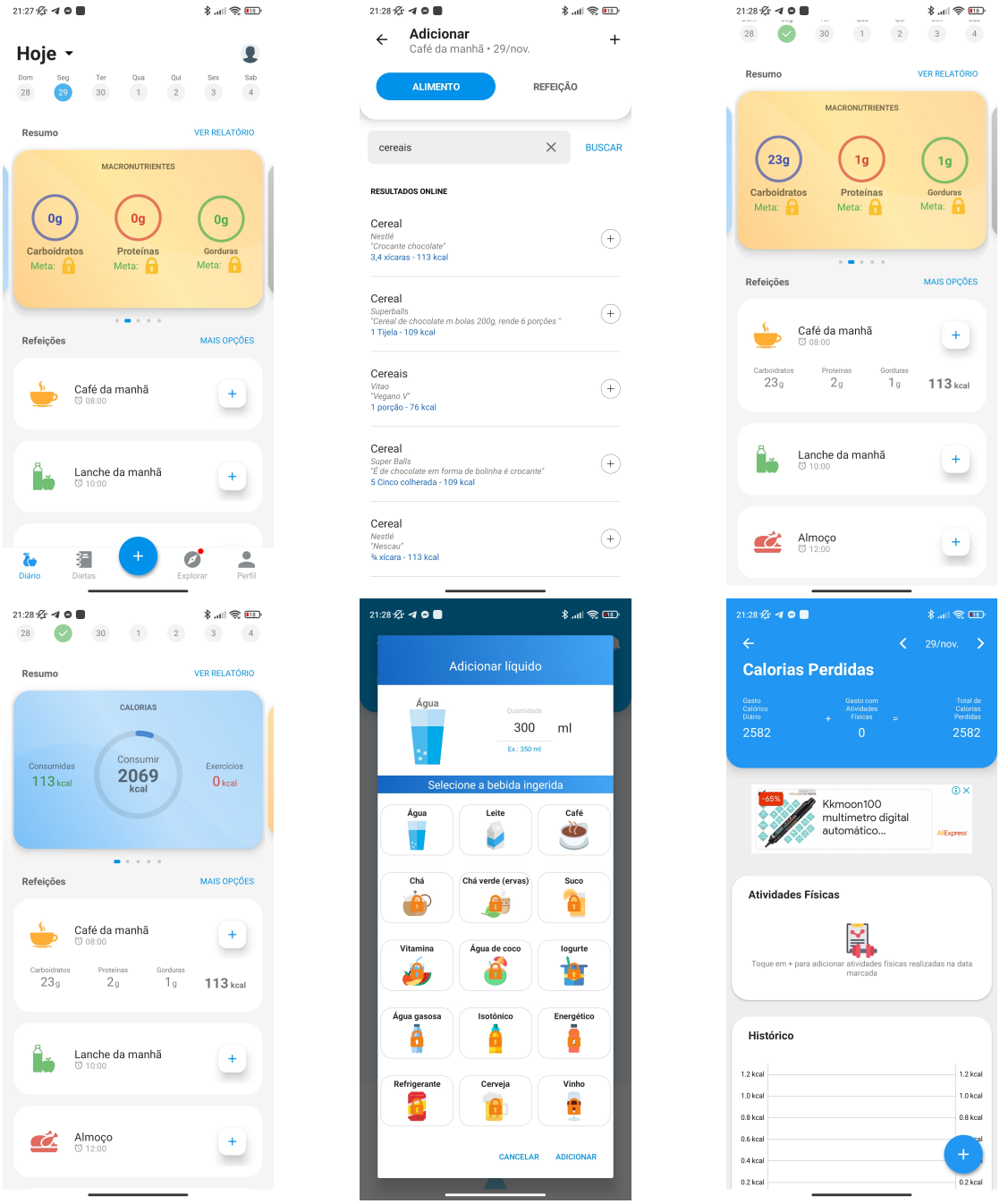



Figure 2.1: *Alimente-se*: representative images of the application's layout.

### 2.2.2 MyFitnessPal

The app begins with a page for creating an account, where it requests information about the user intentions for using the app (whether it's for weight loss or weight gain), target weight, and weekly weight loss goal (ranging from 250g to 1kg per week). Once the registration is complete, a new page appears, providing a daily calorie consumption goal based on the inputs.

#### Features

*MyFitnessPal* offers both free and premium features, as detailed below.

Free:

- Daily food diary with a list of all consumed foods, calculating consumed calories.

Premium:

- Set goals for carbohydrates, proteins, and fats.
- Food analysis, including a scanner that uses the phone's camera to suggest ingredients to add to the diary.
- Recipes for daily meals.

#### Layout

The app's layout features a menu at the bottom of the screen, with all pages presented in a list format. When the user opens the diary, the current day is displayed, and it is possible to navigate to previous days. On this same page, the user can add ingredients to various meals throughout the day (Breakfast, Lunch, Dinner, Snacks). Additional feutures include log exercises performed and track water consumption. In Figure 2.2, we present representative images of the application's layout.

**Ratings**

- Android: Rated 4.5 out of 5 based on 2.5 million reviews.
- iOS: Rated 4.7 out of 5 based on 1.3 million reviews.

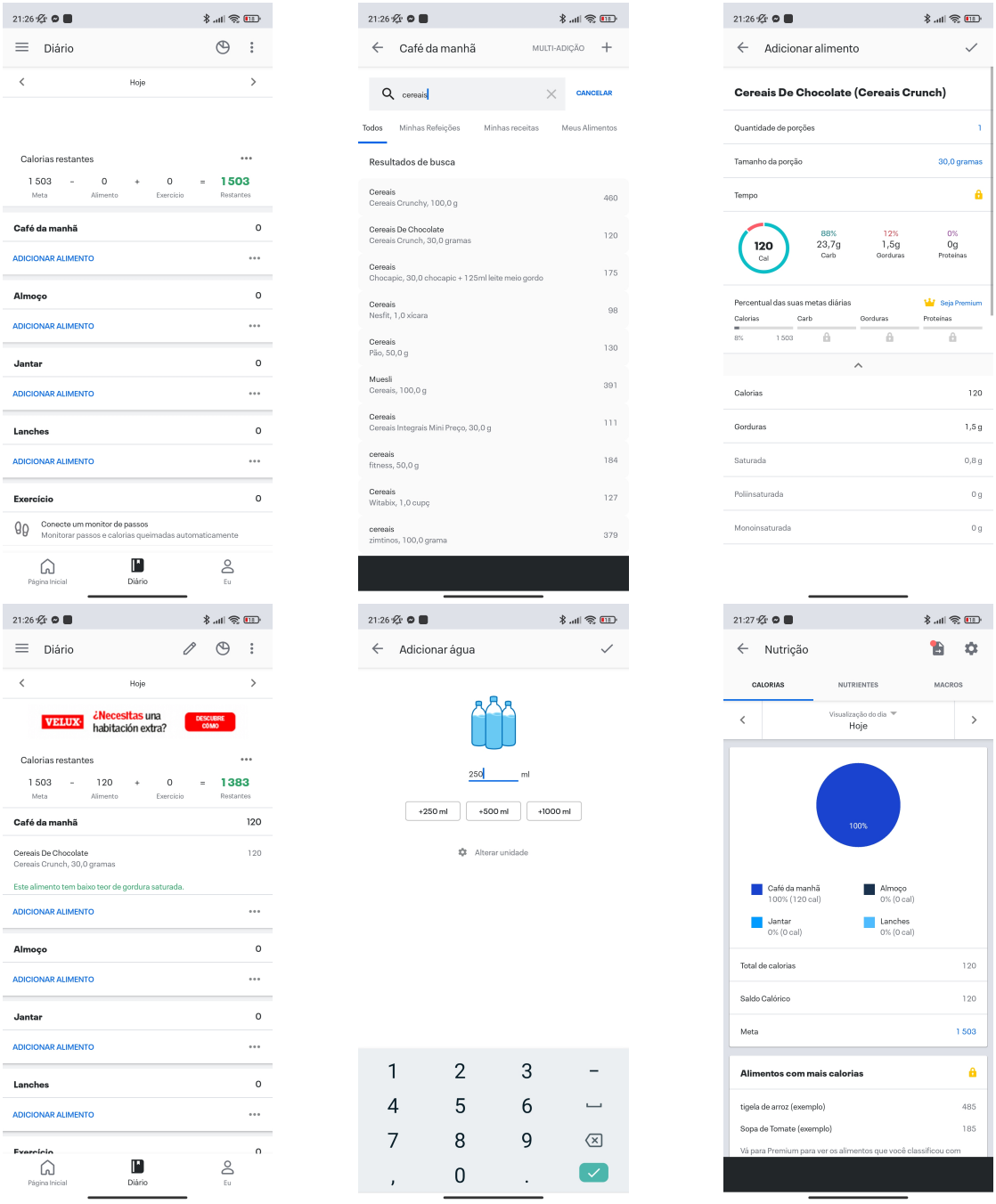



Figure 2.2: *MyFitnessPal*: representative images of the application's layout.

### 2.2.3 Nutrients

In contrast to the previous apps, this app offers a straightforward approach to manage the user dietary choices. Users create their own meal plans, essentially having access to a database to check nutritional information.

#### Features

*Nutrients* offers an user-friendly layout with two main characteristics:

- Viewing nutritional information for each ingredient.
- Creating recipes and having data on the combination of ingredients used.

#### Layout

Very simple layout of a single list where you navigate through categories until you reach a specific food.

#### Rating

This app is only available for iOS where it is rated 4.2 based on 709 reviews.

### 2.2.4 Nutrium

*Nutrium* is a software platform designed for nutritionists and dietitians to manage and optimize their practice. Alongside the platform, the *Nutrium app* is shared by the nutritionists affiliated with their clients. This is an example of a hybrid app that bridges the gap between healthcare professionals and patients.

The information provided is based on the website information. The app is untested as it is restricted to clients of affiliated nutritionists.

#### Features

- Customer data management

- Nutritional and anthropometric assessment
- Predictive equations
- Sharing documents with clients
- Personalizing PDFs with the logo
- Online appointment scheduling
- Integration with video calls
- Appointment reminders
- Physical activity and water analysis
- Chat with clients
- Recording weight in the app
- Real-time tracking
- Food diary with calculations

**Layout**

Like the other apps, it has an layout in the form of a list with all the statistical data and the meal plan.

**Ratings**

Ratings not available.

### 2.2.5 Conclusions

After examining these apps, as well as others in the market, they share a consistent layout. This layout adopts a single-column format that closely resembles a list structure.

This design choice enhances the user experience by simplifying data retrieval within food diaries and the presentation of ingredient and recipe lists.

Furthermore, a notable trend is the uniformity in features offered by these apps. Most prominently, they include a food diary for logging daily dietary intake, provide access to a wide range of recipes, and enable users to monitor the calories burned through physical activity. Notably, features for tracking water intake and recording weight are also widely integrated.

Overall, nutrition apps are highly sought after by customers looking to manage their diet effectively and make informed food choices. Their popularity continues to rise as more people prioritize their health and nutrition. In this way, AquaeVitae's goal of creating a reliable solution designed for nutritionists and patients, responding to the specific needs of each party, is current and relevant.

This chapter summarises, from the used point-of-view, which features we must consider in the design of our solution. Next, in Chapter 3, we present the mobile mockups for our application (Chapter 3.1.6), which have taken into account the research presented here.

# Chapter 3

# Modeling and Technologies

## 3.1 Modeling

At the beginning of our design process, we decided it was best to split the system into two different parts: a web system for nutritionists and administrators, and a mobile one for patients. We planned both parts, but we've only built the web system so far, with the mobile part still being developed separately.

### 3.1.1 Requirements

In our initial discussions about the project's goals, we gathered requirements for each user of the system. These users are the nutritionist, administrator, and patient. The requirements for each are:

**Administrator:**

1. Manage user roles
2. Manage appointment goals
3. Manage pathology types
4. Manage activity levels

5. Manage specificity types
6. Manage food categories
7. Manage food table
8. Manage meal types
9. Manage food-meal associations

**Nutritionist:**

1. Register users
2. Access user personal data
3. Schedule user appointments
4. Specify appointment goals
5. Manage anthropometric data
6. Manage biochemical data
7. Manage user diagnoses
8. Manage user pathologies
9. Add forbidden foods for users
10. Create nutritional plans
11. Specify nutritional plan limits (calories, lipids, proteins, carbs)
12. Specify nutritional plan frequency
13. Manage prohibited foods for plans
14. Define meal amounts and timings in plans

15. Add food options to meals in plans
16. Receive limit exceedance alerts
17. Request automated plan filling
18. Validate/modify automated plans
19. Create meal options with food quantities
20. Access user food diary
21. View patient's historical nutritional plans

**Patient:**

1. Access personal nutritional plan
2. Select consumed foods in meals
3. Report off-plan consumed foods
4. Access personal food diary
5. View nutritional plan statistics
6. Manage food preferences
7. Manage disliked foods
8. Update personal data
9. View appointments
10. Receive appointment notifications
11. View personal diagnostics
12. View personal pathologies

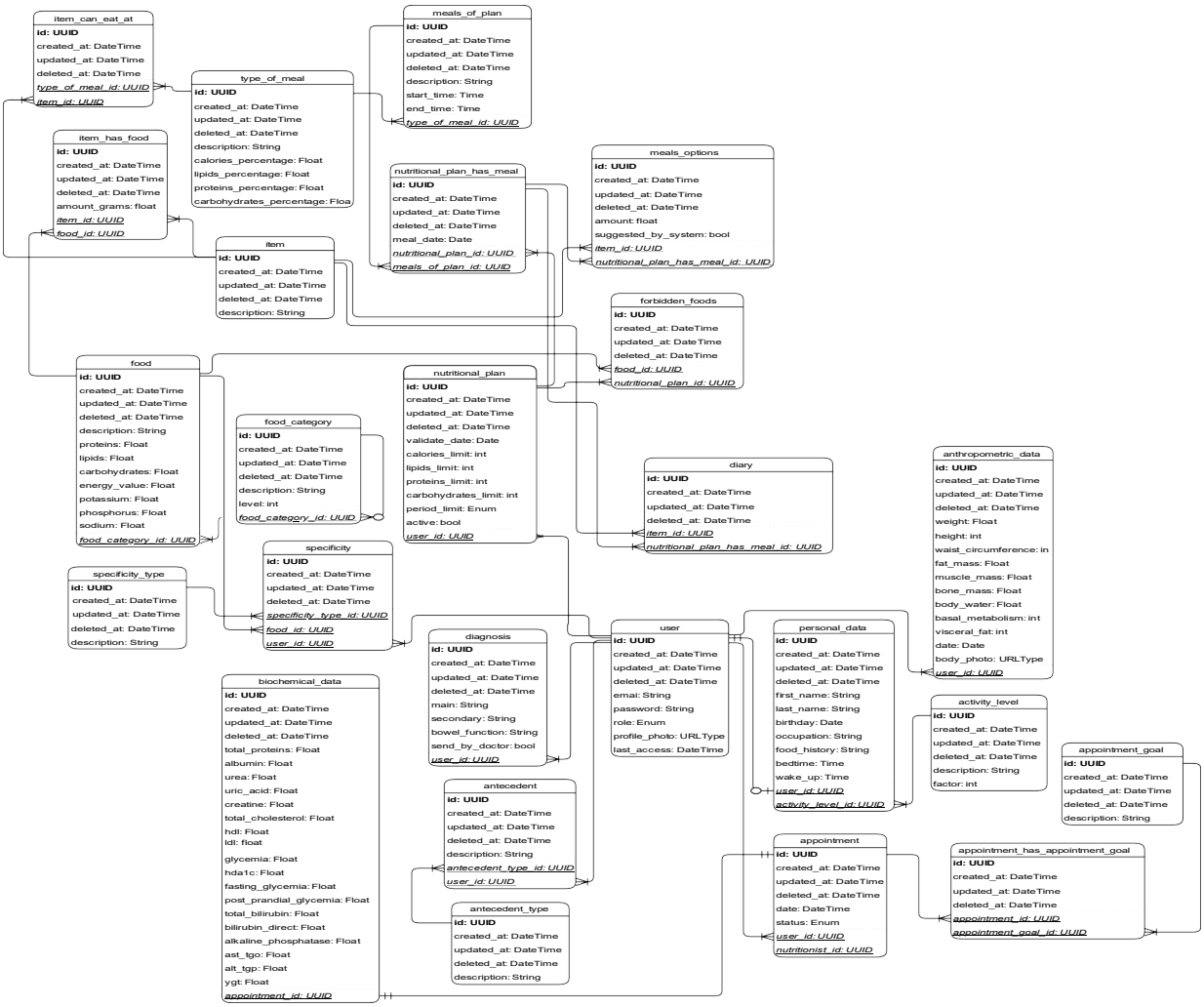
All these functionalities were implemented in the front-end and make use of a set of endpoints developed at backend side.

### 3.1.2 Use Cases

After gathering the initial requirements, we proceeded to create a use case diagram. In this section we show an adaptation of the original use case diagram [4] that had been developed for the backend project. During this adaptation, our focus was on simplifying the diagram by removing elements that were not pertinent to our frontend development. This approach resulted in a diagram that more accurately represents the functionalities and requirements specific to this web application. (Appendix A).

### 3.1.3 ER Diagram

The entity-relationship diagram presented in this section was developed as part of the initial stages of our project. It details the database structure, encompassing entities for storing information about patients, plans, foods, among others. While this diagram was not directly utilized in the frontend development, its creation was a significant time investment during the project's early phase.


item_can_eat_at
id: UUID
created_at: DateTime
updated_at: DateTime
deleted_at: DateTime
type_of_meal_id: UUID
item_id: UUID
type_of_meal
id: UUID
created_at: DateTime
updated_at: DateTime
deleted_at: DateTime
description: String
calories_percentage: Float
lipids_percentage: Float
proteins_percentage: Float
carbohydrates_percentage: Floa
meals_of_plan
id: UUID
created_at: DateTime
updated_at: DateTime
deleted_at: DateTime
description: String
start_time: Time
end_time: Time
type_of_meal_id: UUID
item_has_food
id: UUID
created_at: DateTime
updated_at: DateTime
deleted_at: DateTime
amount_grams: float
item_id: UUID
food_id: UUID
item
id: UUID
created_at: DateTime
updated_at: DateTime
deleted_at: DateTime
description: String
nutritional_plan_has_meal
id: UUID
created_at: DateTime
updated_at: DateTime
deleted_at: DateTime
meal_date: Date
nutritional_plan_id: UUID
meals_of_plan_id: UUID
meals_options
id: UUID
created_at: DateTime
updated_at: DateTime
deleted_at: DateTime
amount: float
suggested_by_system: bool
item_id: UUID
nutritional_plan_has_meal_id: UUID
forbidden_foods
id: UUID
created_at: DateTime
updated_at: DateTime
deleted_at: DateTime
food_id: UUID
nutritional_plan_id: UUID
food
id: UUID
created_at: DateTime
updated_at: DateTime
deleted_at: DateTime
description: String
proteins: Float
lipids: Float
carbohydrates: Float
energy_value: Float
potassium: Float
phosphorus: Float
sodium: Float
food_category_id: UUID
food_category
id: UUID
created_at: DateTime
updated_at: DateTime
deleted_at: DateTime
description: String
level: int
food_category_id: UUID
nutritional_plan
id: UUID
created_at: DateTime
updated_at: DateTime
deleted_at: DateTime
validate_date: Date
calories_limit: int
lipids_limit: int
proteins_limit: int
carbohydrates_limit: int
period_limit: Enum
active: bool
user_id: UUID
diary
id: UUID
created_at: DateTime
updated_at: DateTime
deleted_at: DateTime
item_id: UUID
nutritional_plan_has_meal_id: UUID
anthropometric_data
id: UUID
created_at: DateTime
updated_at: DateTime
deleted_at: DateTime
weight: Float
height: int
waist_circumference: in
fat_mass: Float
muscle_mass: Float
bone_mass: Float
body_water: Float
basal_metabolism: int
visceral_fat: int
date: Date
body_photo: URLType
user_id: UUID
specificity_type
id: UUID
created_at: DateTime
updated_at: DateTime
deleted_at: DateTime
description: String
specificity
id: UUID
created_at: DateTime
updated_at: DateTime
deleted_at: DateTime
specificity_type_id: UUID
food_id: UUID
user_id: UUID
diagnosis
id: UUID
created_at: DateTime
updated_at: DateTime
deleted_at: DateTime
main: String
secondary: String
bowel_function: String
send_by_doctor: bool
user_id: UUID
user
id: UUID
created_at: DateTime
updated_at: DateTime
deleted_at: DateTime
email: String
password: String
role: Enum
profile_photo: URLType
last_access: DateTime
personal_data
id: UUID
created_at: DateTime
updated_at: DateTime
deleted_at: DateTime
first_name: String
last_name: String
birthday: Date
occupation: String
food_history: String
bedtime: Time
wake_up: Time
user_id: UUID
activity_level_id: UUID
activity_level
id: UUID
created_at: DateTime
updated_at: DateTime
deleted_at: DateTime
description: String
factor: int
appointment_goal
id: UUID
created_at: DateTime
updated_at: DateTime
deleted_at: DateTime
description: String
biochemical_data
id: UUID
created_at: DateTime
updated_at: DateTime
deleted_at: DateTime
total_proteins: Float
albumin: Float
urea: Float
uric_acid: Float
creatine: Float
total_cholesterol: Float
hdl: Float
ldl: float
glycemia: Float
hda1c: Float
fasting_glycemia: Float
post_prandial_glycemia: Float
total_bilirubin: Float
bilirubin_direct: Float
alkaline_phosphatase: Float
ast_tgo: Float
alt_tgp: Float
ygt: Float
appointment_id: UUID
antecedent
id: UUID
created_at: DateTime
updated_at: DateTime
deleted_at: DateTime
description: String
antecedent_type_id: UUID
user_id: UUID
antecedent_type
id: UUID
created_at: DateTime
updated_at: DateTime
deleted_at: DateTime
description: String
appointment
id: UUID
created_at: DateTime
updated_at: DateTime
deleted_at: DateTime
date: DateTime
status: Enum
user_id: UUID
nutritionist_id: UUID
appointment_has_appointment_goal
id: UUID
created_at: DateTime
updated_at: DateTime
deleted_at: DateTime
appointment_id: UUID
appointment_goal_id: UUID

### 3.1.4 Architecture Diagram

Below we present the Architecture diagram of the overall project, color-coded to highlight our contribution in the different aspects. In green, we highlight our core focus and in yellow the areas mainly developed by other project members were we have intervened. This distinction in color coding in the diagram helps to delineate our direct contributions and the extent of our involvement in different parts of the total project.

In the scope of this report, we mainly focus on the development of the Web application for both Nutritionists and Administrators (highlighted in green). This has received our full attention in terms of development and analysis.

On the other hand, the sections highlighted in yellow were extensively covered in the backend project [4]. Nevertheless, we have initiated the planning of the yellow-marked components and, in the final stage, conducted the required adjustments in line with the final version of the Web Application.

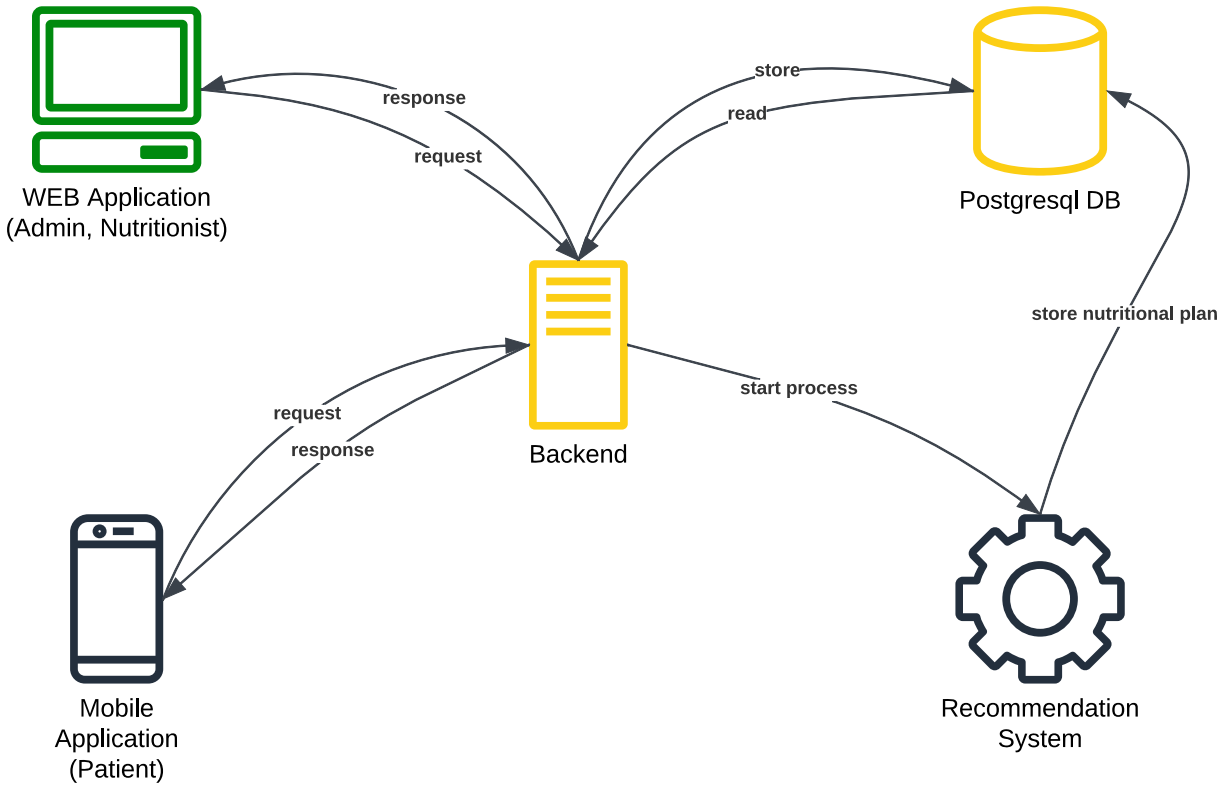
While the backend project is not the central focus of this current report, we decided to present it due to the contributions provided in the early and late phases of the project.

### 3.1.5 WEB Mockups

For the web system, we used a layout that has a side menu with options for the nutritionist. There's also a top bar displaying the current user's information with a menu to access their profile and log out. We chose this layout because we believe it's intuitive for users, especially since it's commonly used for admin pages.

In Figure 3.1, there's an example of a list layout, which we'll use on most pages containing lists. Specifically, this figure shows the page listing foods. We can see a table displaying some food details, and right above it, there's a button to access the food creation form. The search bar on the top right lets us filter foods by their description.

Figure 3.2 displays the list of patients. Specifically, this is one of the pages where we won't use the layout from Figure 3.1. We can observe that each patient is represented by a box containing their photo, first and last name, and date of birth.



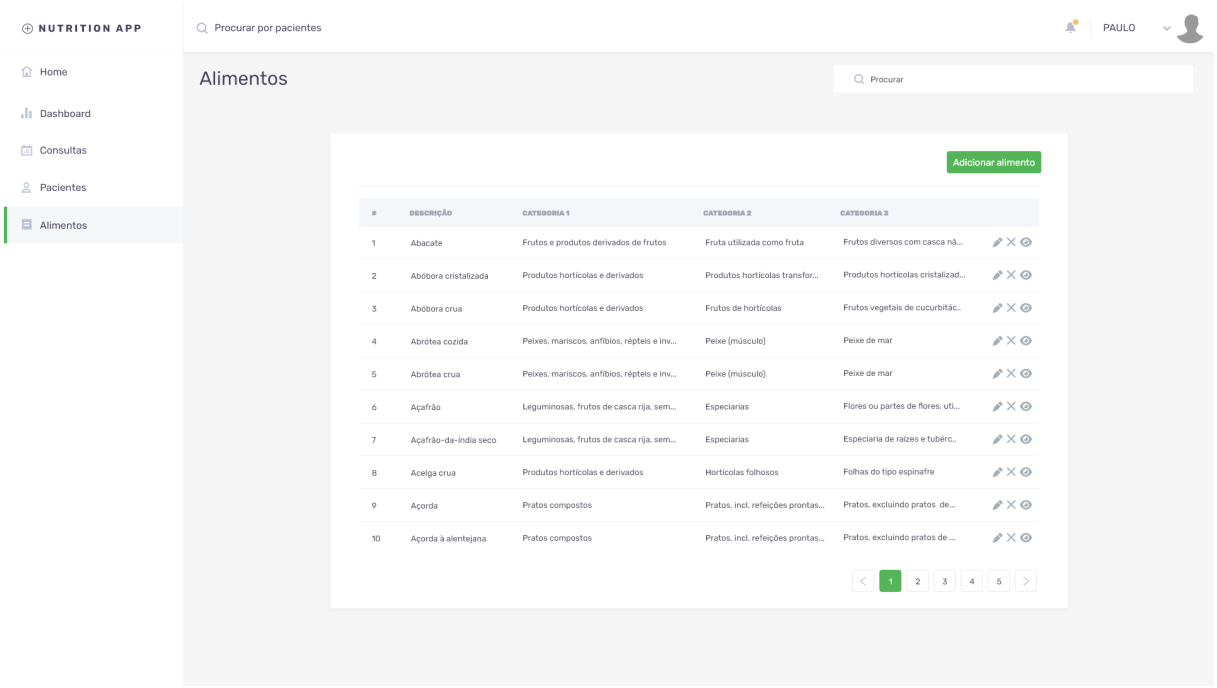




Figure 3.1: List Example

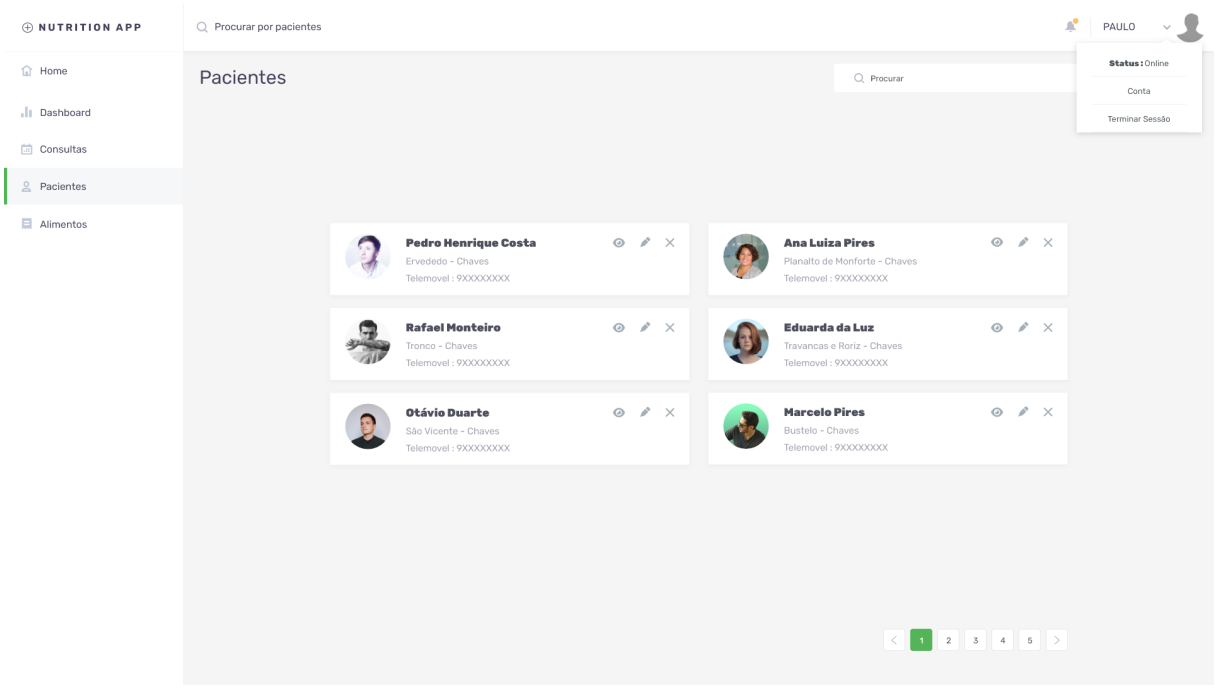



Figure 3.2: Patient List

The mockup for the nutritional page was one of the most crucial ones during our initial discussions to ensure it was very intuitive for the nutritionist. As depicted in Figure 3.3, it consists of various sections ranging from the total micronutrients target, the number of meals, to the list of foods to be consumed for each meal.

In Figure 3.4, we can see an example of a form. All the forms in our system will look quite similar to each other. In this specific example, the form is divided into multiple sections because it's one of the more complex ones. This division helps maintain simplicity and intuitiveness, which would be lost if users were presented with an overly lengthy form.

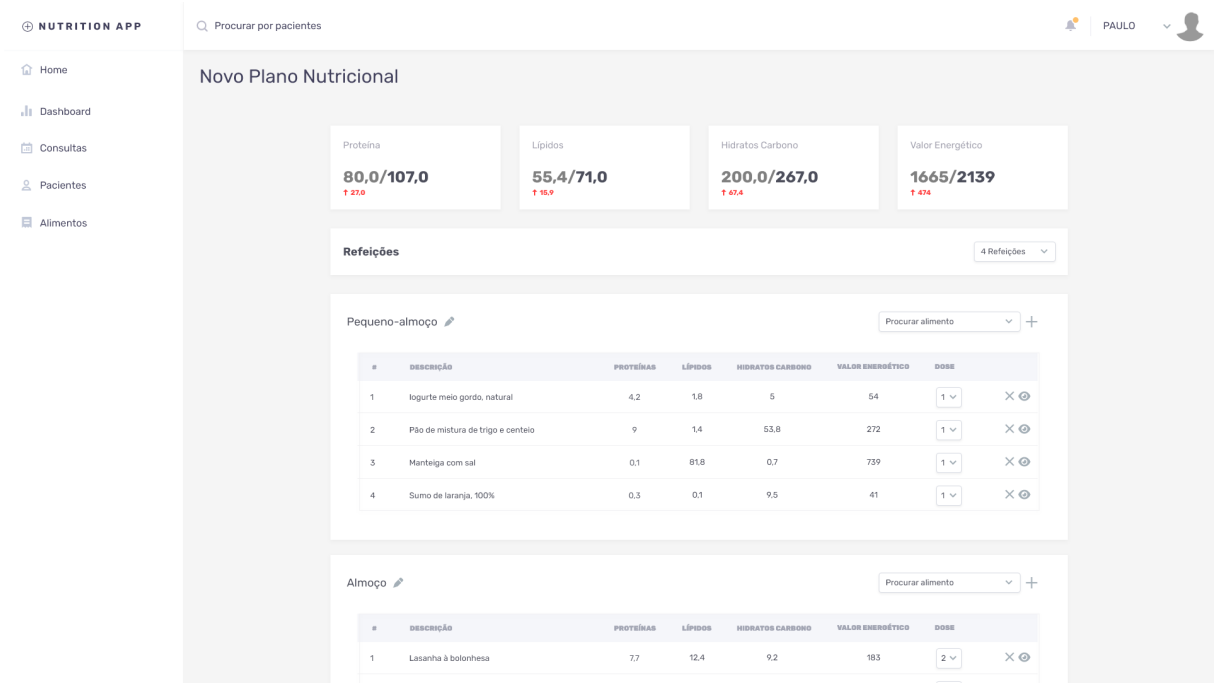



Figure 3.3: Nutritional Plan Page

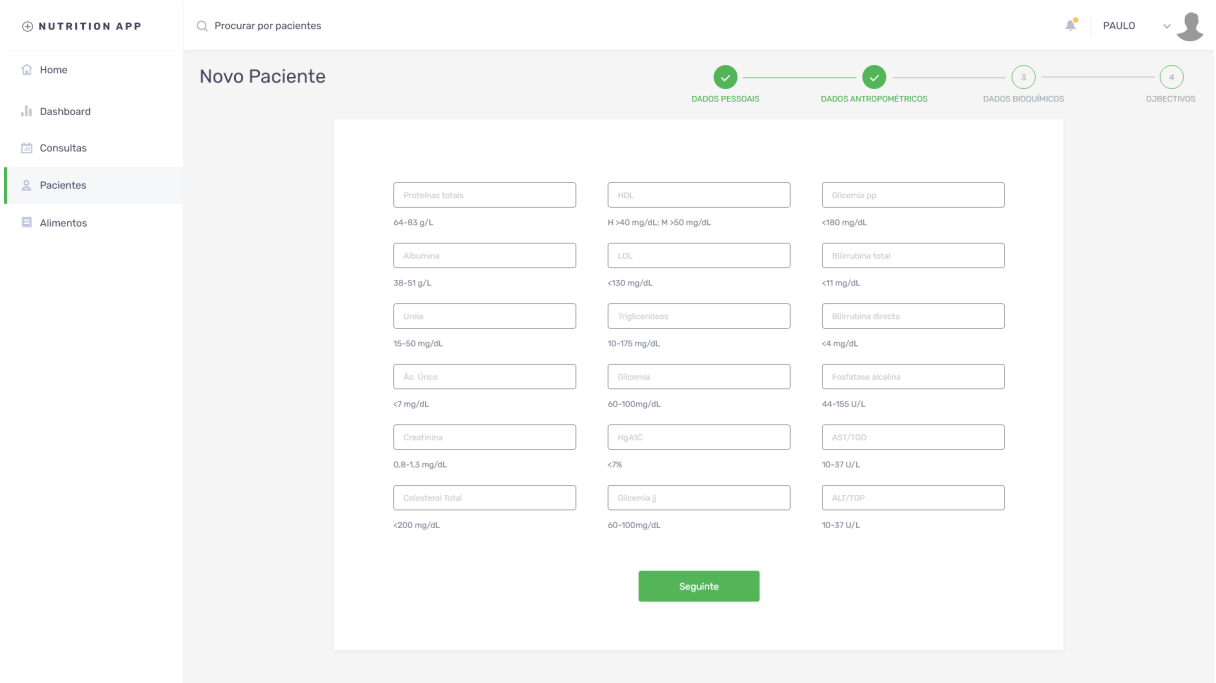



Figure 3.4: Form example

### 3.1.6 Mobile Mockups

The mobile mockups for our application were designed using Adobe XD. These mockups encapsulate the main functionalities of the system, illustrating a conceptual vision for the mobile app layout. It's important to note that these mockups represent a separate project, they are conceptual designs that are currently in implementation. Furthermore, the creation of some of these mockups was inspired by the information detailed in Chapter 2. This inspiration is evident in the way the mockups align with the system's requirements and objectives as laid out in that chapter, providing a visual representation of the potential mobile app experience.

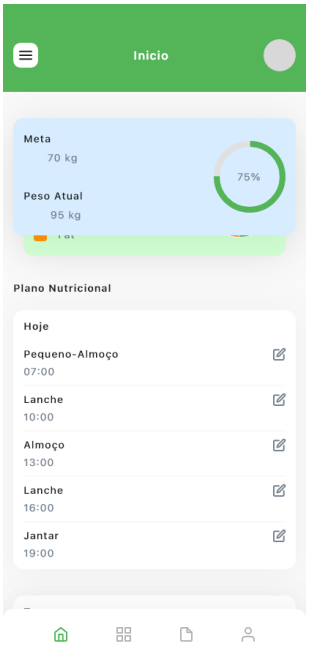



Figure 3.5: Home


Figure 3.6: Nutritional


Figure 3.7: Diary

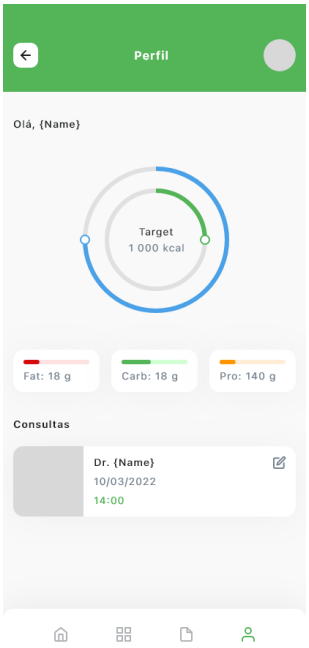



Figure 3.8: Profile


Figure 3.9: Results

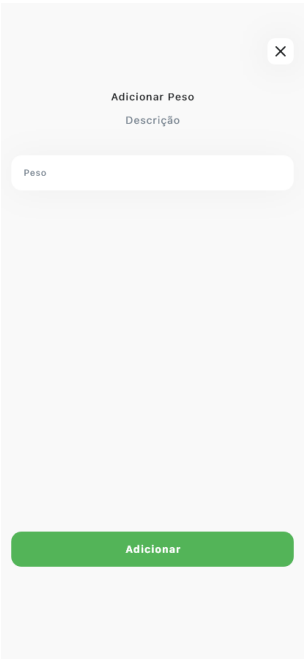



Figure 3.10: Form example

### 3.1.7 Application Programming Interface

In this section, we will get into the API developed as part of the backend project [4]. Our focus will be on providing a concise yet informative description of several key aspects of the API. By highlighting these important aspects, we aim to give a clear understanding of how the API facilitates communication and data exchange within our application, and how it integrates with other components of the system.

**User Authentication**

1. Login

   Path: /login

   Method: POST

   Description: Authenticates a user and returns login details.

2. Current User

   Path: /me

   Method: GET

   Description: Retrieves the currently authenticated user's details.

**User Management**

1. Create User

   Path: /user/create

   Method: POST

   Description: Allows for the creation of a new user.

2. Get All Users

   Path: /user/get

   Method: GET

   Description: Retrieves a list of all users, with pagination support.

**CRUD (Create, Read, Update, Delete)**

1. Create

   Path: /entity-path/create

   Method: POST

   Parameters: Includes params, and other relevant fields.

   Response: Returns the details of the created entity.

2. Read

   Path: /entity-path/get

   Method: GET

   Parameters: May include filters like name or id.

   Response: Provides a list of entities with pagination.

3. Update

   Path: /entity-path/update/id

   Method: PATCH

   Parameters: Entity id and fields to update.

   Response: Confirmation of the update with the updated details.

4. Delete

   Path: /entity-path/delete/id

   Method: DELETE

   Parameters: Enitity id

   Response: Confirmation of deletion.

The API uses OAuth2PasswordBearer for security, ensuring that endpoints require proper authentication for access.

The responses of this API are well-structured, providing detailed information about the success or failure (validation errors) of requests. Most endpoints return JSON data corresponding to the specific operation, like user details, activity level information, etc.

## 3.2 Technologies

In this chapter the technologies used to develop the web application are presented.

### 3.2.1 Typescript

TypeScript, developed by Microsoft, is a statically typed superset of JavaScript, enhancing code reliability and maintainability. It's popular in web development for building larger, more complex applications [5] [6], and it is suitable for a wide range of projects due to its compatibility with JavaScript and advanced features [7].

For JavaScript developers, TypeScript offers an easy transition, allowing seamless integration into existing projects. Moreover, this is a strategic choice for enhancing JavaScript projects with more robust and maintainable code [7].

TypeScript static typing allows developers to specify types, reducing runtime errors. This technology improves the development experience with features like autocompletion and code refactoring, essential for managing large codebases. TypeScript has strong support from major tech companies and a growing community, ensuring continuous updates and resources [6]. Overall, these features are beneficial for maintaining code quality in large projects [8]

TypeScript is suitable for a wide range of projects due to its compatibility with JavaScript and advanced features, TypeScript is a strategic choice for enhancing JavaScript projects with more robust and maintainable code [7].

### 3.2.2 Node.js

Node.js is an open-source, cross-platform, back-end JavaScript runtime environment that runs on the V8 engine and executes JavaScript code outside a web browser. It is known for its non-blocking, event-driven architecture, making it efficient for building scalable network applications [9].

Node.js simplifies the development process by utilizing JavaScript for both client-side and server-side scripting [10]. Since this cross-platform executes JavaScript on the server side, it allows building entire web applications in JavaScript. It is particularly renowned for its speed and efficiency, due in large part to its non-blocking I/O and asynchronous event-driven processing [11].

This technology is widely used for developing scalable and high-performance applications, particularly real-time web applications (like chat servers and collaborative tools) and REST APIs [11] [10]. Its package manager, npm, boasts one of the largest ecosystems of open-source libraries [12].

The Node.js community is active in the software world and offers a vast array of tools, libraries, and frameworks. Its ecosystem is ever-growing, thanks to regular updates and contributions from a large pool of developers [13].

### 3.2.3 Vue.js

Vue.js is a progressive JavaScript framework used for building user interfaces and single-page applications. It's known for its ease of integration, scalability, and gentle learning curve [14].

Vue.js is an excellent choice for web developers seeking a framework with a flexible architecture, in a JavaScript framework landscape [15] [16]. This framework emphasizes a declarative and component-based programming model. It allows for the efficient creation of dynamic user interfaces, with its reactive data binding and composable view components [15] [14].

From a learning perspective, Vue.js is approachable for those with basic HTML, CSS,

and JavaScript knowledge. Its simplicity and detailed documentation make it accessible to beginners, while its robustness appeals to experienced developers [16]. The development experience is improved by the clear and concise syntax, a supportive community, and a rich ecosystem of supporting libraries and tools [14] [17].

Vue.js boasts a strong community and ecosystem, including tools like Vuex for state management and Vue Router for SPA routing. It's well-suited for both small projects and large-scale enterprise applications [17] [18].

### Choosing Vue.js

In the development of our project, we carefully considered various technologies to ensure the most efficient and effective solution. Among these, we chose Vue.js as our primary framework for building the user interface. Our decision was influenced by several key factors, which set Vue.js apart from other popular frameworks like React.js and Angular.

One of the primary reasons we opted for Vue.js over React.js or Angular is our team's familiarity and extensive experience with Vue.js. This experience translates into faster development times, reduced learning curves, and a more streamlined development process. Vue.js's comprehensive documentation and supportive community further aid in quick problem-solving and learning, making it an ideal choice for our project needs.

Also Vue.js is known for its lightweight nature and optimal performance. Compared to React.js and Angular, Vue.js has a smaller bundle size, leading to faster load times and enhanced performance - a critical factor for our project's user experience. Additionally, Vue.js's virtual DOM implementation and reactive data binding system ensure a high level of efficiency, particularly in handling dynamic content, which is a significant aspect of our project.

Vue.js offers a high degree of flexibility, allowing us to structure our project as we see fit. This adaptability is particularly useful in accommodating the unique requirements of our project. Furthermore, the Vue.js ecosystem, including tools like Pinia for state management and Vue Router for page navigation, provides us with a comprehensive set of functionalities, making the development process more seamless and integrated.

In conclusion, our decision to use Vue.js for our project was driven by its ease of integration, enhanced development experience due to our team's familiarity with the framework, performance benefits, and its flexible yet powerful ecosystem. These attributes of Vue.js align perfectly with our project goals and requirements, making it a superior choice over React.js and Angular for our specific needs.

### 3.2.4 Axios

Axios is a popular JavaScript library used to make HTTP requests from node.js or XML-HttpRequests from the browser, with a promise-based nature, and the ability to handle requests and responses in JSON format [19].

Axios has comprehensive documentation that provides clear examples and guides, making it accessible to beginners. For those familiar with JavaScript and HTTP request concepts, Axios is relatively easy to learn [19]. In addition, Axios benefits from a robust community and is an open-source project with regular updates and contributions from developers across the globe [20].

For developers needing to make HTTP requests in their web applications, Axios is an excellent tool. This dependency includes a wide range of features, including the ability to make HTTP GET and POST requests, support for synchronous and asynchronous requests, and automatic transformation of request and response data. Moreover, this library offers interceptors to manipulate requests and responses before handling them. Its simplicity and features make it a preferred choice for API interactions in JavaScript applications [21].

Using Axios in development projects simplifies the process of working with APIs. Its promise-based structure integrates seamlessly with modern JavaScript frameworks like React, Vue.js, and Angular, enhancing the development workflow [21].

### 3.2.5 Quasar Framework

The Quasar Framework is a high-performance, full-front-end stack that improves the development of Vue.js applications. It allows for the building of responsive websites, PWAs (Progressive Web Apps), mobile apps (through Cordova or Capacitor), and Electron apps using the same codebase [22].

Quasar is known for its wide range of integrated features and tools. It provides a rich set of UI components, ensuring consistency and responsiveness across different platforms and devices. This framework is especially beneficial for projects requiring a single codebase for multiple platforms. Quasar enhances the development experience by offering a CLI (Command Line Interface) for quick project scaffolding, an extensive component library, and automatic code splitting for improved app performance. It also integrates well with popular development tools and plugins.

The Quasar Framework is an open-source tool in constant evolution. It represents an excellent choice for developers looking to leverage Vue.js for building cross-platform applications. Its comprehensive feature set, ease of use, and strong community support make it a compelling choice for modern web and mobile development [22] [23].

## 3.3 Conclusions

The Aquavitae system's design includes key features common to diet management systems, while also integrating innovative aspects. These include:

- **Patient Feedback System:** An interactive tool allowing patients to regularly provide feedback about their diets, enhancing communication and real-time diet plan adjustments.
- **Automated Diet Generation:** The system employs a recommendation system to automatically create personalized diets based on user needs and preferences.
- **Nutritionist Monitoring Report:** Nutritionists can access detailed reports on patient progress, improving the diet monitoring and adjustment process.

- **New Foods in the Food Database:** Particularly important for the AquaeVitae project is the addition of new thermal-based foods developed within the project itself. This enriches the system's food database, offering a broader range of diet options.

These features combine to make AquaeVitae not just a diet management tool, but a platform for health and wellness, focusing on innovative interactions between nutritionists and patients and the introduction of advanced nutritional elements.

# Chapter 4

# Implementation

In this chapter, the implementation of the web application is described. All the code described in this chapter is available on GitHub Repository [24].

## 4.1 Structure

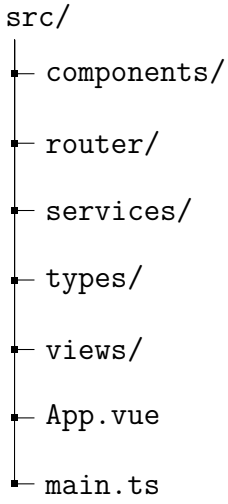



Figure 4.1: Project Structure

To keep a project clean and easily understood, it's crucial to organize it well. This means sorting project files into different folders or categories. In the following sections of this chapter, we'll show various file examples and explain their purposes for each of the folders shown in Figure 4.1.

## 4.2 Router

By default, Vue.js doesn't support navigation routes without added dependencies. To address this, most developers use vue-router. This allows us to link routes to different views and pass properties to these views via the URL. In Example 4.1, we see how a route is defined. It has several elements:

- **path**: The navigation route used to access the view.
- **name**: An internal name for dynamic navigation within the project's code.
- **component**: The imported view file. (For better performance, we only import it when the route is needed).
- **props**: This determines if URL properties are passed as props to the component. These properties are set in the path with a ":" before the variable name we want to use.

```
1 {
2   path: '/menu-user/nutrition-plan/view/:user_id',
3   name: 'nutrition-plan',
4   component: () => import('@/views/patient/details/nutrition/view.vue'),
5   props: true,
6 }
```

Listing 4.1: Route Example

## 4.3 Services

Services are files that handle requests to an API. In this folder, we have a file for almost every entity in our database, except those not relevant to the web application. Each file typically has at least four functions, which are the basic CRUD (Create, Read, Update, Delete) operations. Below, we'll provide examples (4.2, 4.3, 4.4, 4.5, 4.6) of each of these functions and explain their purpose.

The index function (4.2) retrieves a list of appointment goals from an API, and it allows for detailed customization on how the list is fetched:

- **page** and **itemsPerPage**: Define which set of data items to retrieve, useful for pagination.
- **sort**: Determines the order in which the items are returned (default is by description in ascending order).
- **columns**: Specifies which columns or fields from the data to retrieve. Can be a single column (like 'description') or multiple columns.
- **filter**: Allows for filtering the results based on certain criteria.

Inside the function:

A new URLSearchParams object is created to hold the parameters for the API request. It sets the desired columns, page, number of items, sorting method, and any search filters to this object.

Depending on whether columns and filter are arrays or single values, it either loops through each item and appends it or appends the single value directly.

The function then calls the get method from the API with the defined parameters to retrieve the desired list of appointment goals.

The function was designed to be as versatile as possible. This allowed us to build components like 4.6.3, where all the services are dynamic, a topic we'll discuss later in this section.

```
1 index(
2   page: number,
3   itemsPerPage: number,
4   sort = 'description:ASC',
5   columns = 'description' as string | string[],
6   filter = null as string | null
7 ) {
8   const params = new URLSearchParams()
```

```
9
10    if (Array.isArray(columns)) {
11      for (let column of columns) {
12        params.append('columns', column)
13      }
14    } else {
15      params.append('columns', columns)
16    }
17    params.append('skip', page.toString())
18    params.append('take', itemsPerPage.toString())
19    params.append('sort', sort)
20    if (filter) {
21      if (Array.isArray(filter)) {
22        for (const fil of filter) {
23          params.append('search', fil)
24        }
25      } else {
26        params.append('search', filter)
27      }
28    }
29
30    return Api().get(`appointment-goal/get`, {
31      params,
32    })
33  },
```

Listing 4.2: Service Method Get All Example

The show function (4.3) retrieves the details of an appointment goal based on its id.

```
1  show(id: string) {
2    return Api().get(`appointment-goal/get/${id}`)
3  },
```

Listing 4.3: Service Method Get One Example

The post function (4.4) creates a new appointment goal using the provided appointmentGoal object.

```
post(appointmentGoal: AppointmentGoal) {
  return Api().post('appointment-goal/create', appointmentGoal, {
    headers: {
      'Content-Type': 'application/json',
    },
  })
},
```

Listing 4.4: Service Method Post Example

The put function (4.5) updates an existing appointment goal using its id and the updated appointmentGoal object

```
put(appointmentGoal: AppointmentGoal) {
  return Api().patch(
    `appointment-goal/update/${appointmentGoal.id}`,
    appointmentGoal,
    {
      headers: {
        'Content-Type': 'application/json',
      },
    }
  )
},
```

Listing 4.5: Service Method Put Example

The delete function (4.6) removes an appointment goal based on its id.

```
delete(id: number) {
  return Api().delete(`appointment-goal/delete/${id}`)
},
```

Listing 4.6: Service Method Delete Example

## 4.4 Types

In a Vue.js project, the types folder plays a important role when integrating TypeScript, a popular static type checker. This directory houses TypeScript type definitions, interfaces, and sometimes enums, ensuring type safety across the application. By defining and centralizing these types, we can achieve better code clarity, catch potential errors during compile-time, and enhance the overall maintainability of the Vue application. The types folder acts as a reference point for the structure and expected data types of components, props, events, and state. In essence, it's a important quality for projects robustness and scalability in a type-safe environment.

```
type Appointment = {
  id?: string
  created_at?: string
  updated_at?: string
  deleted_at?: string
  status: AppointmentStatus
  user: User
}
```

Listing 4.7: Type Definition

## 4.5 Views

In Vue.js development, the views folder holds a special significance. This directory typically contains Vue components that represent the different views or pages of the application. Unlike the components in the components folder, which are often smaller and reusable pieces of the UI, the components within the views folder are more extensive. Each file in the views folder corresponds to a distinct route in the application, they serve as the primary layout for a particular route or page. In essence, the views folder acts as the central hub where top-level layouts reside, making it easier to manage and evolve the application's page structure over time.

```
<template>
  <List
    title="Objetivos"
    type="AppointmentGoal"
    add-button
    add-button-text="Adicionar Objetivo"
    table-sort-by="description"
    filter-column="description"
    :columns="columns"
    create-form="appointment-goals-create-form"
  >
  </List>
</template>
<script lang="ts">
import { defineComponent } from 'vue'

import List from '@/components/misc/List.vue'

export default defineComponent({
  components: {
    List,
  },
  data() {
    return {
      columns: [
        {
          name: 'description',
          label: 'Descricao',
          field: 'description',
        },
      ],
    }
  },
})
```

```
35 </script>
```

Listing 4.8: View Example

## 4.6 Components

### 4.6.1 Base

In the "Base" component, detailed in Appendix B, we've established what can be aptly termed the foundation for all our views and even certain other components. The HTML structure of this component is bifurcated into four distinct sections, with at most three being visible simultaneously. These are:

- Header with title and breadcrumb (Figure 4.2)
- Header without title and breadcrumb (Figure 4.3)
- Header right content
- Main Content

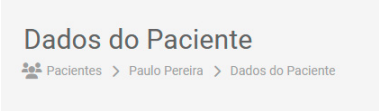



Figure 4.2: Header with Title

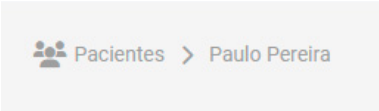



Figure 4.3: Header Without Title

Both the Header Right Content and Main Content essentially serve as slots when the component is in use (see examples 4.9 and 4.10).

```
1 <template #right-header>
2   <search
3     v-model="filter"
4     @click-button="add"
5     :add-button="addButton"
6     :button-text="addButtonText"
```

```
  ></search>
</template>
```

Listing 4.9: Example Usage of Slot Header Right Content

```
<template #content>
  <patient-box
    v-for="user in users"
    :user="user"
    :personal-data="user.personalData"
    @view="view"
    @edit="edit"
    @delete="delete"
    @open-settings="openSettings"
    style="margin: 16px"
  ></patient-box>
</template>
```

Listing 4.10: Example Usage of Slot Main Content

When leveraging this component, we have incorporated a few props. Some are mandatory while others aren't. They include:

**title:** Page title.

**vertical-alignment:** Position of the main content on the vertical axis.

**horizontal-alignment:** Position of the main content on the horizontal axis.

**no-header:** A flag to activate or deactivate the page title.

**direction:** Orientation in which the content is displayed (row or column).

**breadcrumb:** An array detailing the breadcrumb options.

### 4.6.2 Table

The 'Table' Component, Appendix C, create a dynamic and interactive table (showed in figure 4.4) with sorting and pagination features. Here's a breakdown of each part:

- **Table Header:** It includes sortable columns with sorting indicated by arrows changing direction based on the sorting order.
- **Loading State:** When the loading property is set to true, skeleton loaders are displayed for each row, indicating that data is being loaded.
- **Table Rows:** This feature displays data rows where each cell's content can be customized using slots. Text alignment and flex sizing in each cell are dynamically set based on the columns property.
- **No Data State:** If there is no data to display (and it's not in a loading state), a default message ('Sem resultados') is shown to inform users.
- **Pagination:** The table uses Quasar's q-pagination component to facilitate navigation through table pages.

In this section, we'll detail the pagination, sort, columns, and rows props, which are crucial for the dynamics and flexibility of the table component:

**Pagination**

- Manages table pagination.
- Includes sub-properties like *page*, *rowsPerPage*, *pagesNumber*, and *rowsNumber*.
  - *page:* Controls the current page of the table.
  - *rowsPerPage:* Sets the number of lines displayed per page.
  - *pagesNumber:* Total available pages, calculated based on the total number of rows and rows per page.
  - *rowsNumber:* Total number of rows in the table.
- This property enhances user experience in tables with large datasets by allowing page navigation and control over the number of displayed items.

**Sort**

- Manages the ordering of table columns.
- Key sub-properties include *by* and *descending*.
  - *by:* Determines which column is used for sorting.
  - *descending:* A boolean indicating if sorting is descending (true) or ascending (false).
- Enables users to sort table data by clicking on columns, making it easier to find specific information.

**Columns**

- Defines the columns of the table.
- Each array item is an object representing a column, which may include properties like *label* (column label), *name* (unique column identifier), *align* (text alignment), *size* (column size), and others.
- Crucial for customizing the table, it defines what data is displayed and how it is presented.

**Rows**

- Represents the data to be displayed in the table.
- Each array item is an object corresponding to a table row, with keys matching the column names.
- This property is dynamic, allowing the table to display different data sets.

These properties make the table component highly flexible and adaptable to various use cases. With pagination and sort, users can easily navigate and organize large data sets. Columns and rows allow for extensive customization of data presentation.

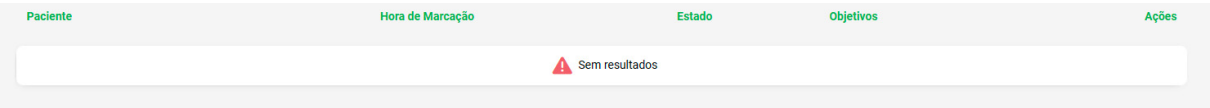



Figure 4.4: Table Component

### 4.6.3 List

We made a component that's easy to use and flexible. It lets us add, change, or remove items. This component connects with a backend service for various data tasks like creating, reading, updating, and deleting. Connected with component table it has features for filtering and sorting data, which makes it easier and better for users to handle data. This component is designed to be adaptable with slots and dynamic properties, so we can use it in different situations. We also included pagination, which is really helpful for handling lots of data. It organizes the data well and makes it easier to go through. We made sure the component is responsive and user-friendly. We wanted it to be smooth for users to interact with. Overall, our goal was to make this component powerful but also simple and nice to use.

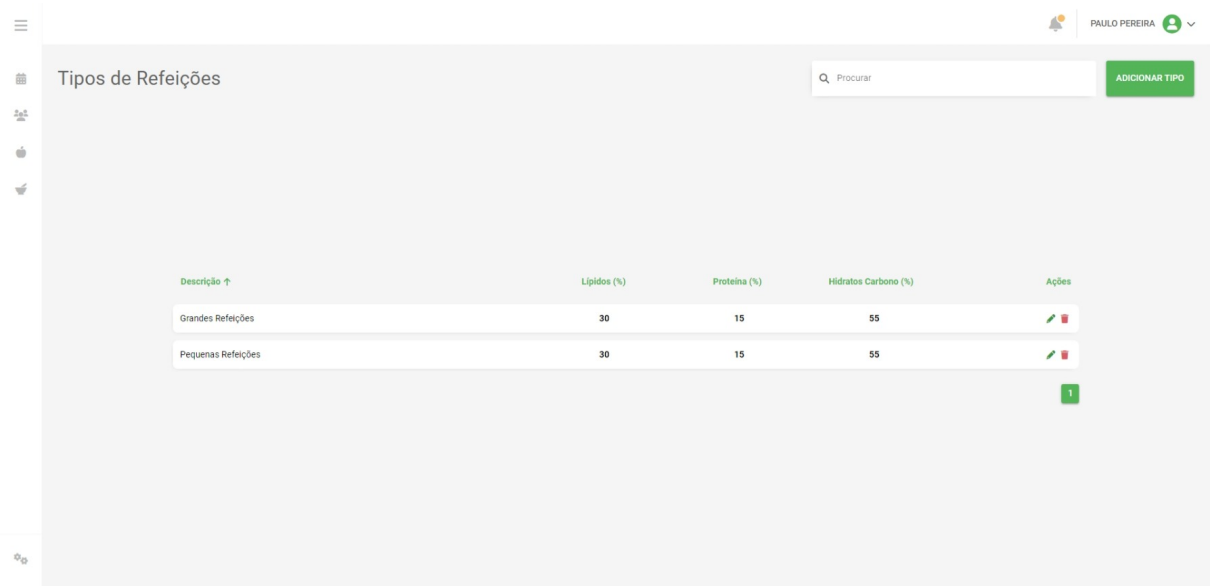



Figure 4.5: List Component

### 4.6.4 Autocomplete

We would use this component in our application where a select dropdown with features like dynamic loading, filtering, and virtual scrolling is needed, typically in scenarios with large datasets or where data is fetched from a server. The props options like valueKey, labelKey, and label make it versatile for different data structures and UI requirements.

In component code (appendix D), we can see that as we scroll through the list, our service continuously monitors our position, when we approach the end of the loaded data, it automatically sends a request to our service to fetch the next set of data. This process is so smooth that we likely won't even notice it happening, yet we'll always have access to more data with a simple scroll.
This brings several benefits, such as:

- **Efficiency in Data Handling:** By loading data on-demand, our application ensures that we only process the data we need at any given moment. This approach is resource-efficient, reducing unnecessary load on our devices and servers.

- **Improved User Experience:** We no longer have to endure long wait times for all data to load before we can start interacting with the application. The data loads progressively, ensuring a smooth, uninterrupted experience.

- **Scalability:** This method is incredibly scalable. Whether we're dealing with hundreds of records or millions, the application maintains its performance, adapting to the volume of data seamlessly.

- **Reduced Bandwidth Usage:** For those of us on limited data plans, this approach minimizes data consumption, as only a fraction of the data is transmitted over the network at a time.

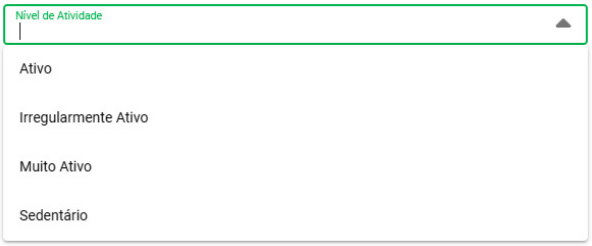



Figure 4.6: Autocomplete

### 4.6.5 Conclusion

In this chapter, we delve into several key components of our project to demonstrate their versatility and scalability. Understanding these components is crucial as they form the backbone of our entire operation and are fundamental in achieving our project goals.

By exploring the services, types, and views of our project, this chapter aims to provide a comprehensive understanding of its versatile and scalable nature. Each component plays a significant role in maintaining the project's coherence and ensuring its success. As we move forward, it's these components' ability to adapt and scale that will guide our project's evolution and success in meeting its objectives.

# Chapter 5

# Results

In this chapter, we'll show the results of our project by giving some examples of how our application turned out. We'll explain the features we added, what the user interface looks like. These examples will help demonstrate how well our development approach worked and the quality of the app we made.

## 5.1 Developed Work

In Figure 5.1, we can see the list of appointments, which is filtered by day and includes a quick access feature for creating new appointments. The appointments are sorted by time and display key details such as patient information, the time of the appointment, its current status, and the goals that the patient has set for that specific appointment.

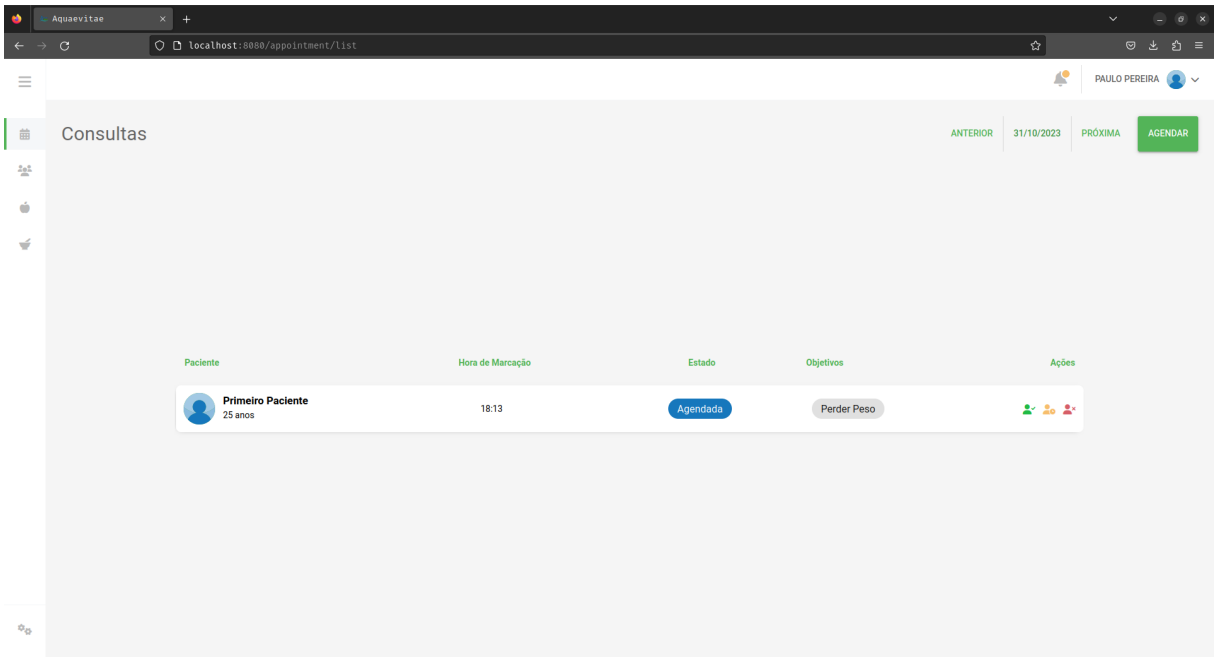



Figure 5.1: Appointment List

In Figures 5.2, 5.3, 5.4, and 5.5, we present a detailed overview of our application's functionality. Figure 5.2 illustrates the list of registered patients, serving as the initial point of interaction within the application. Moving forward, Figure 5.3 depicts the individualized menu for each patient. This menu is a pivotal feature as it facilitates the access to specific patient information.

Particularly, Figure 5.3 is instrumental in navigating to detailed sections such as the anthropometric data of the patients, which is showcased in Figure 5.4. This section is crucial for understanding the physical characteristics and measurements of the patients, vital for tailoring personalized nutritional plans.

Lastly, Figure 5.5 is dedicated to displaying the nutritional plan assigned to each patient. This feature not only outlines the dietary recommendations but also plays a significant role in the overall treatment strategy.

These figures collectively demonstrate the application's ability to manage and present complex patient data in an organized and user-friendly manner, highlighting the efficiency and effectiveness of the system in aiding nutritional management and planning.

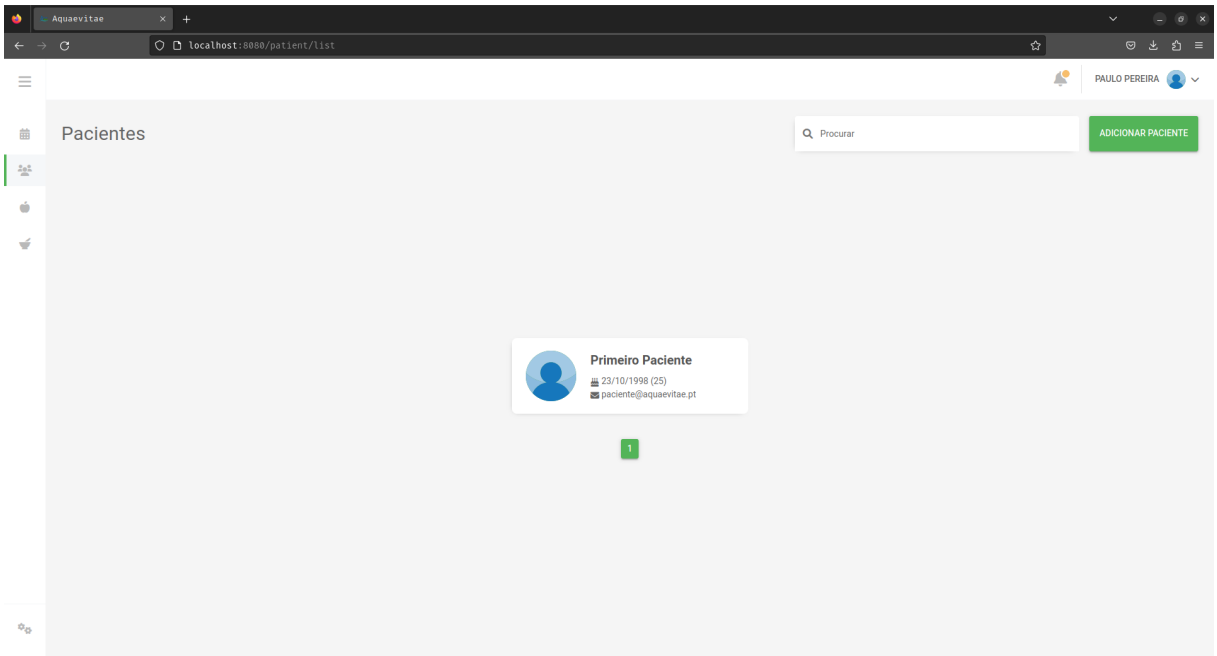



Figure 5.2: Patient List

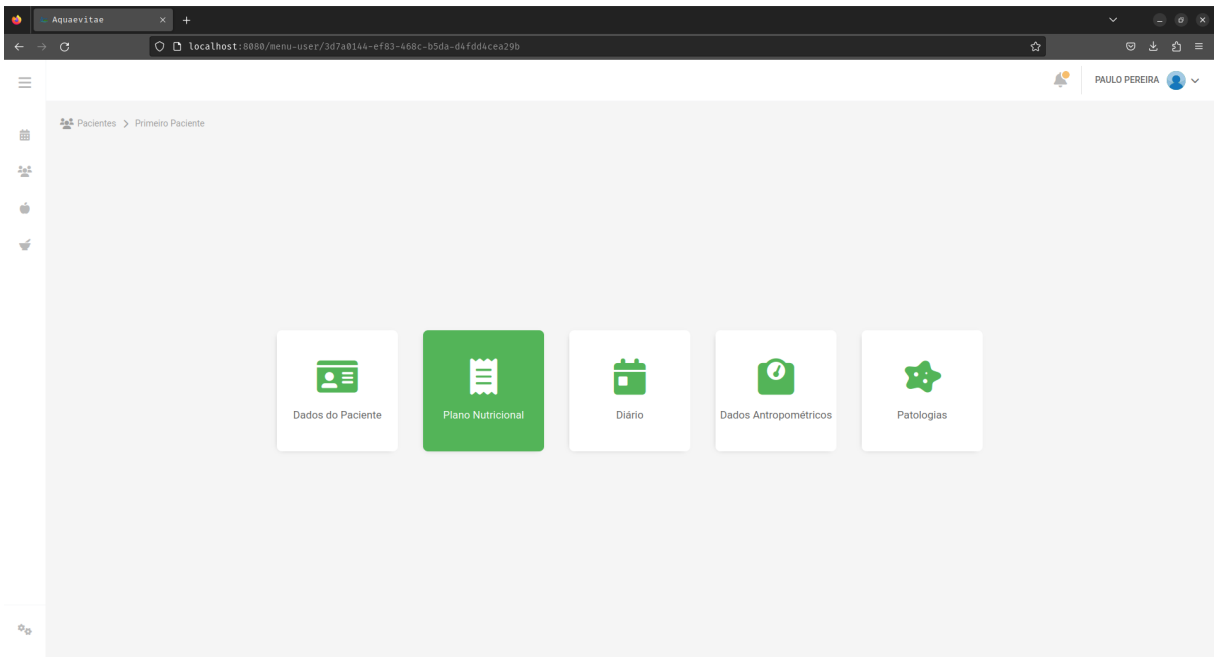



Figure 5.3: Patient Menu

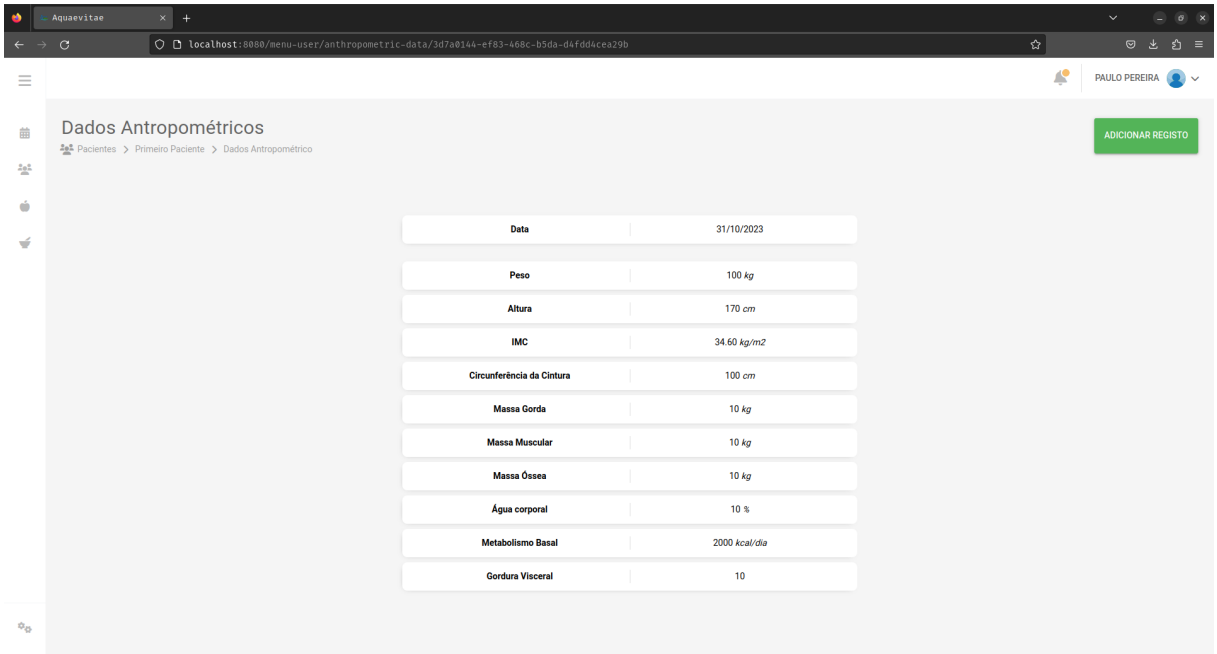



Figure 5.4: Anthropometric Data

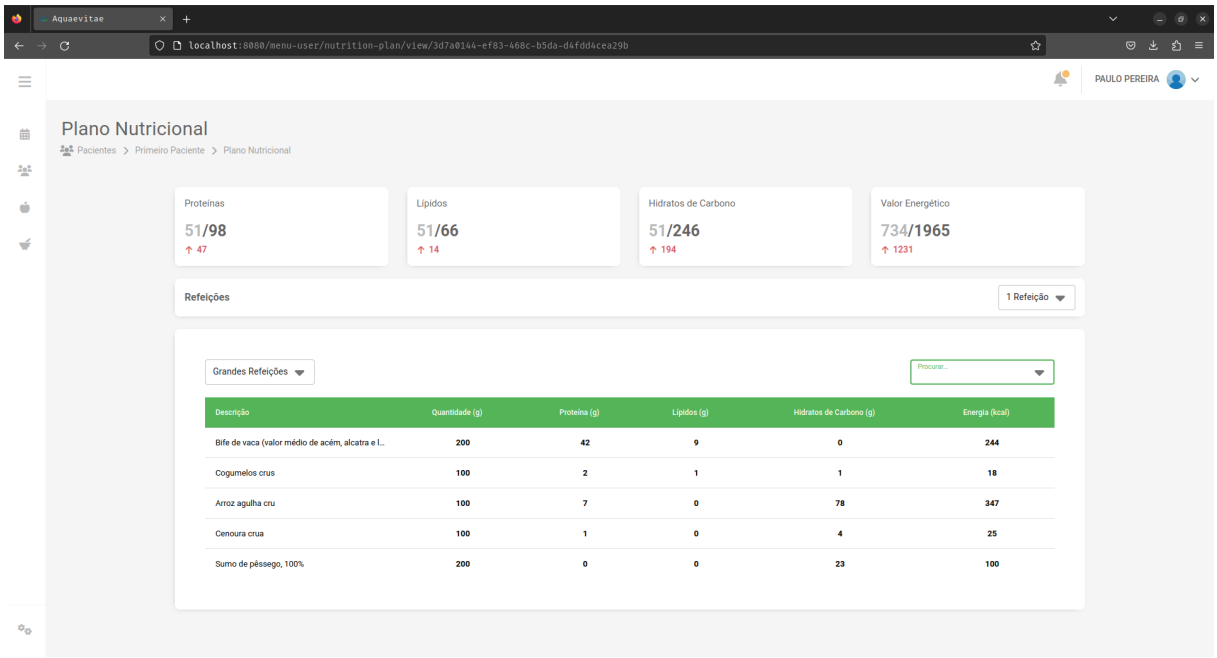



Figure 5.5: Nutritional Plan

In Figures 5.6, 5.7, and 5.8, we illustrate the perspective of a nutritionist regarding the available food items and meals to make nutritional plans. These figures provide a clear view of the user interface that nutritionists interact with when selecting and organizing dietary components for their patients' meal plans. This visual representation underscores the functionality and user-friendly design of the application, highlighting how it aids nutritionists in efficiently managing and creating personalized nutritional strategies.

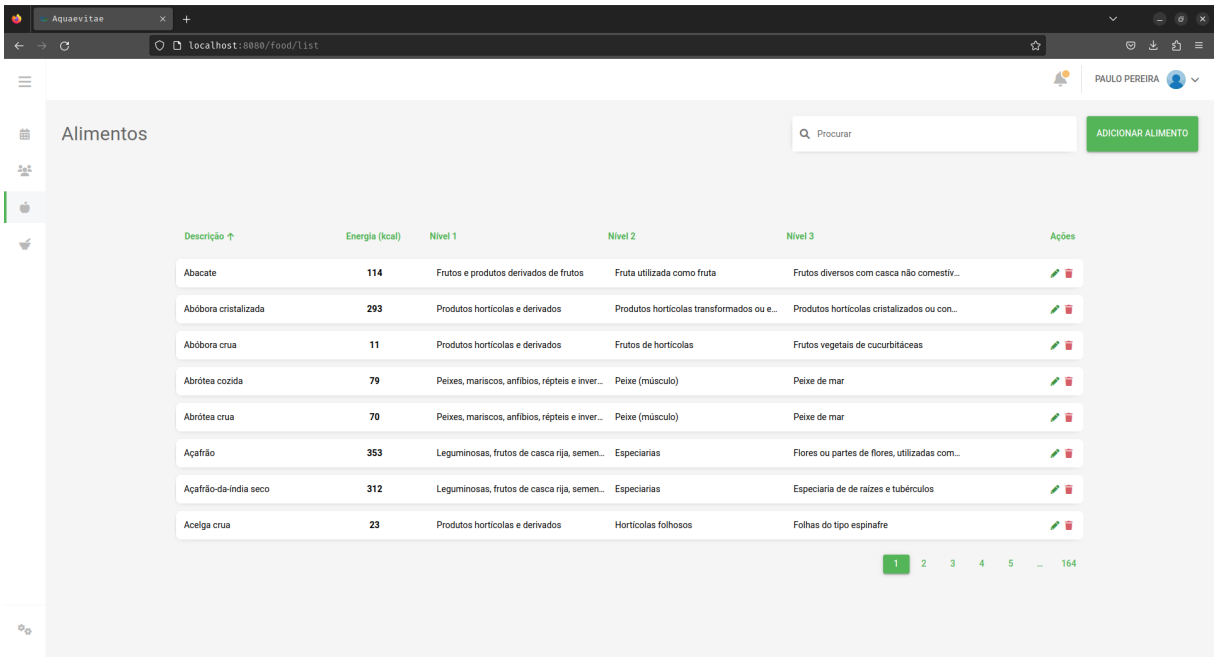



Figure 5.6: Food List

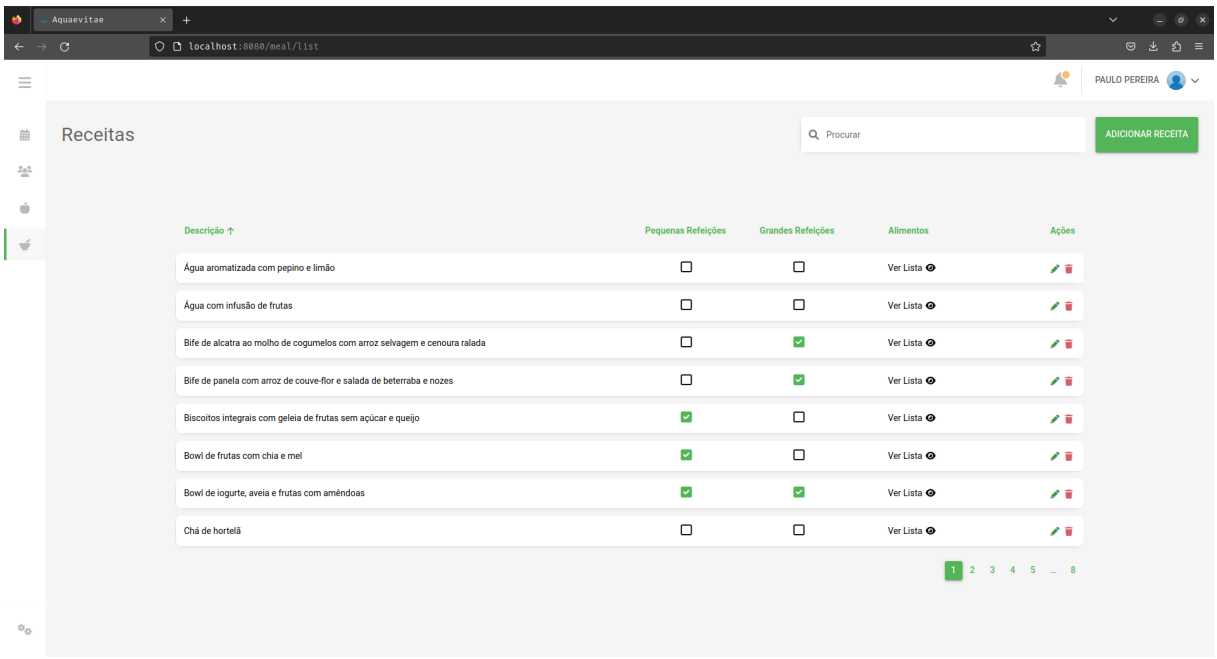



Figure 5.7: Meal List

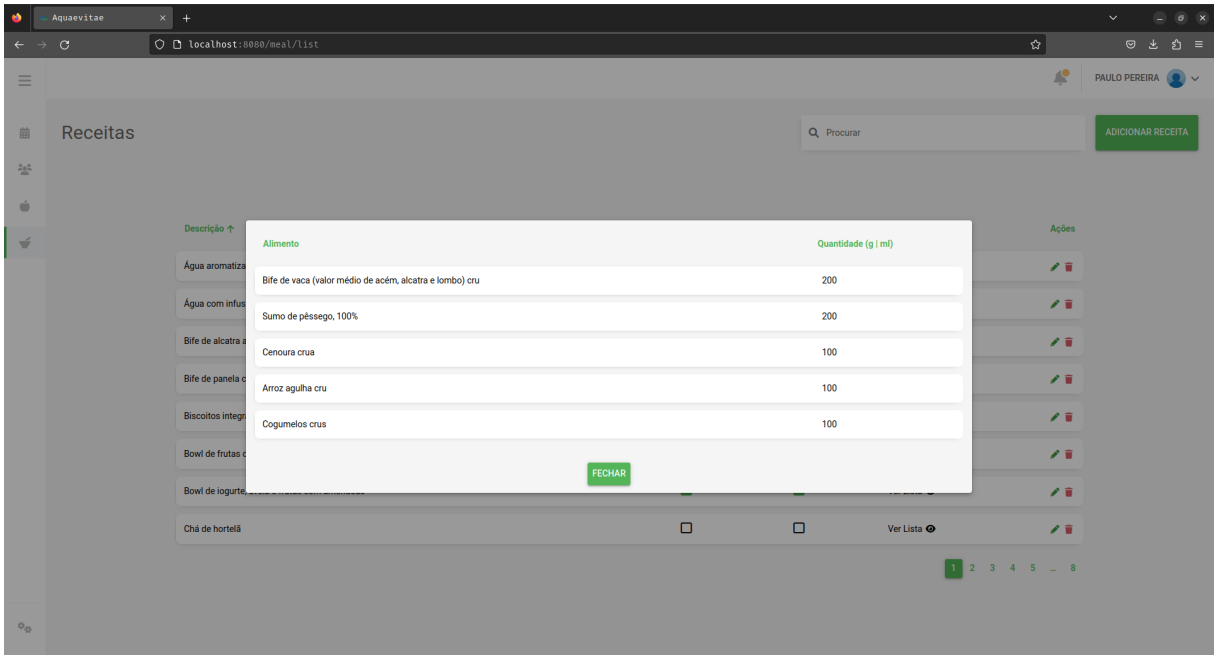



Figure 5.8: Meal Detail Dialog

In Figures 5.9, 5.10, 5.11, and 5.12, we showcase what constitutes the management area of our application. This section is designed for administrators and nutritionists, providing them with the capability to perform various administrative tasks. Here, they can add new users, including administrators and nutritionists, and create different types of meals, among other functionalities. This area is integral to the application as it allows for the efficient management and customization of the system to suit the specific needs of the organization and its clients.

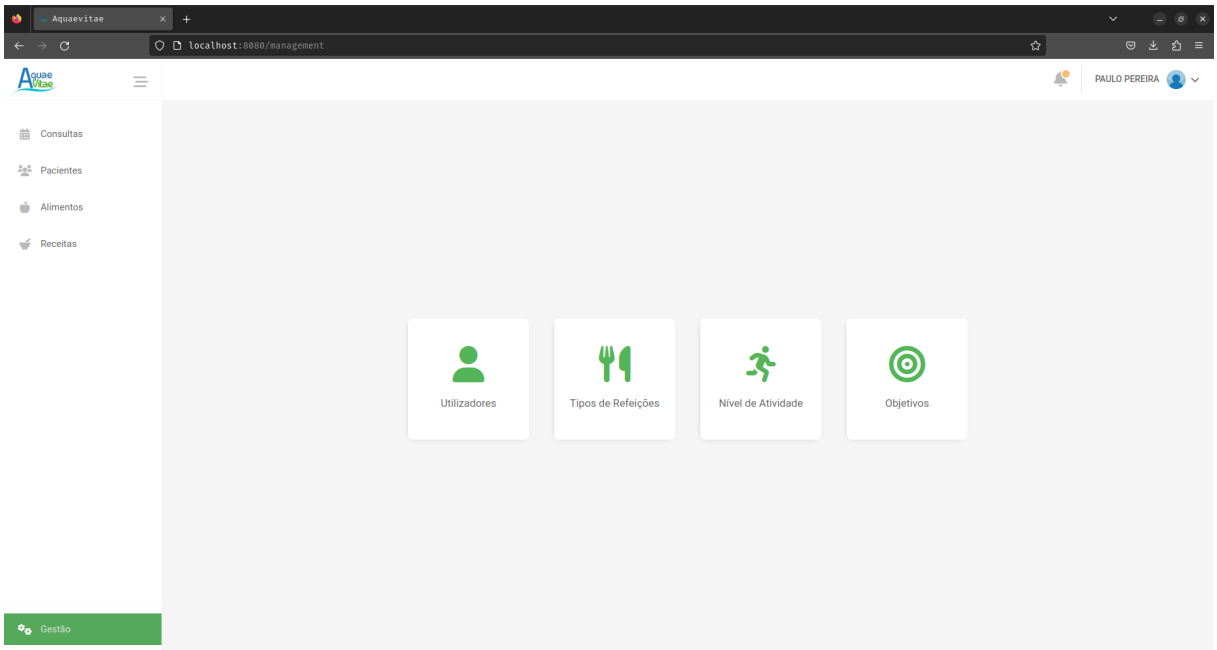



Figure 5.9: Managment Menu

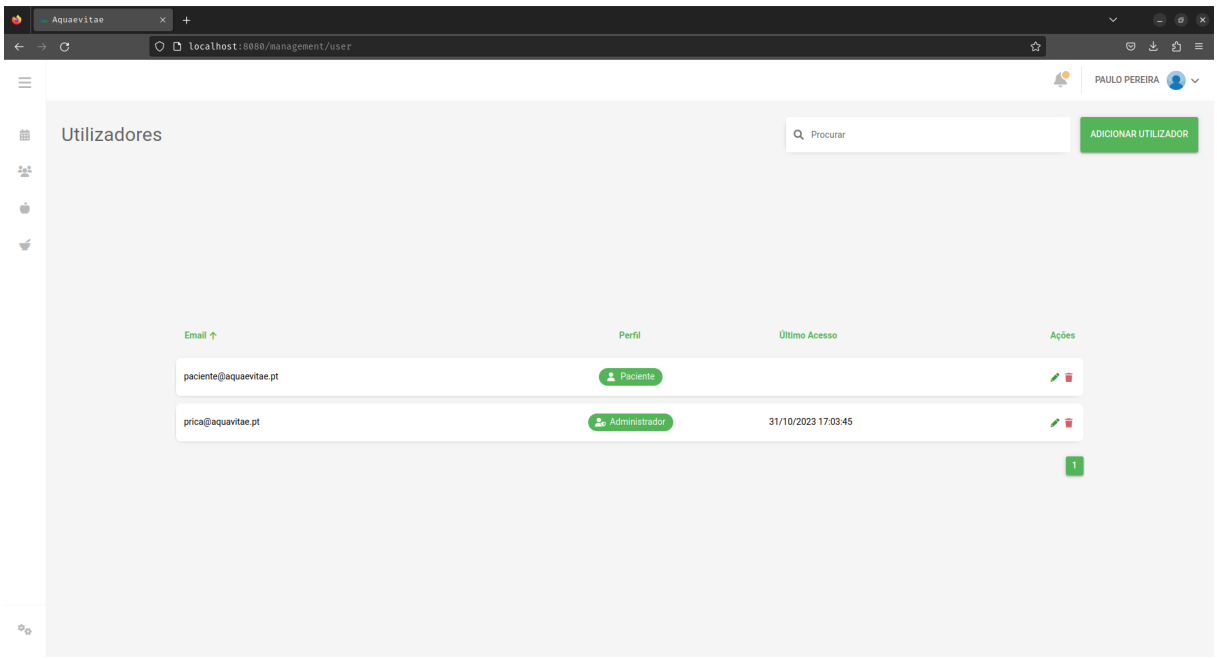



Figure 5.10: All Users List

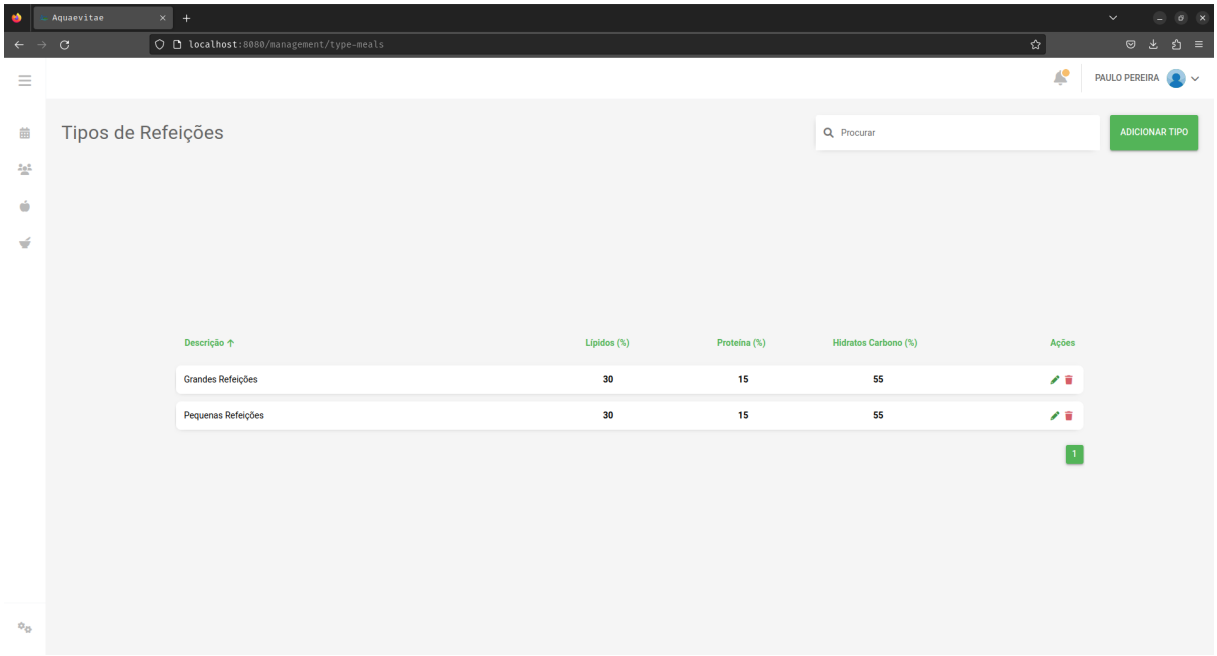



Figure 5.11: Type of Meals List

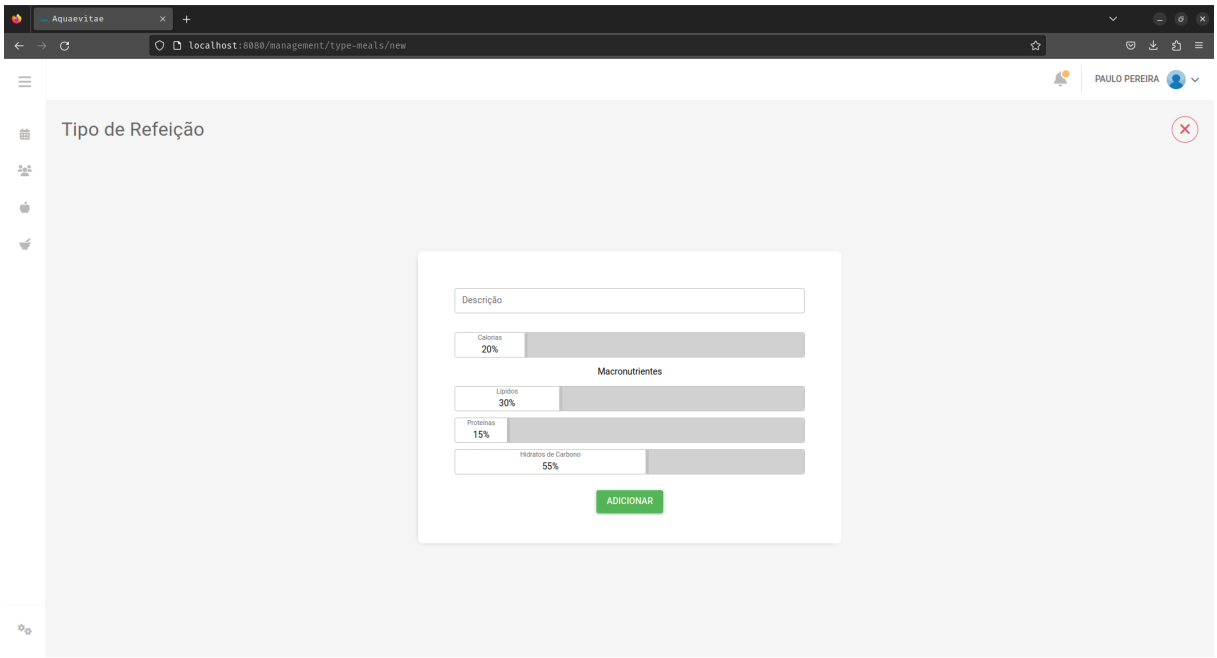



Figure 5.12: Type of Meals Form

The patient record is a complex form in our application. To manage this complexity, we've divided it into several steps, as demonstrated in Figures 5.13, 5.14, and 5.15. This step-wise approach was initially planned and outlined in our mockups. By breaking down the form into more manageable sections, we've made it user-friendly while ensuring comprehensive data collection for each patient's record. This design choice not only improves the user experience but also enhances the accuracy and efficiency of data entry.

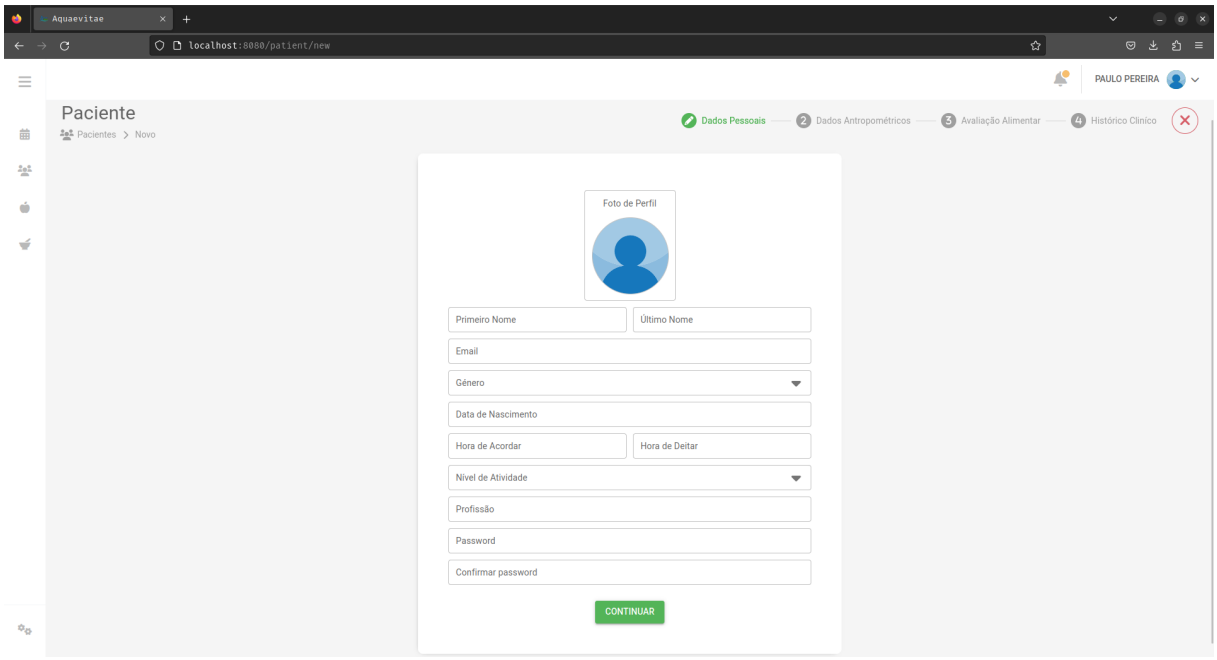



Figure 5.13: Patient File Form Example 1

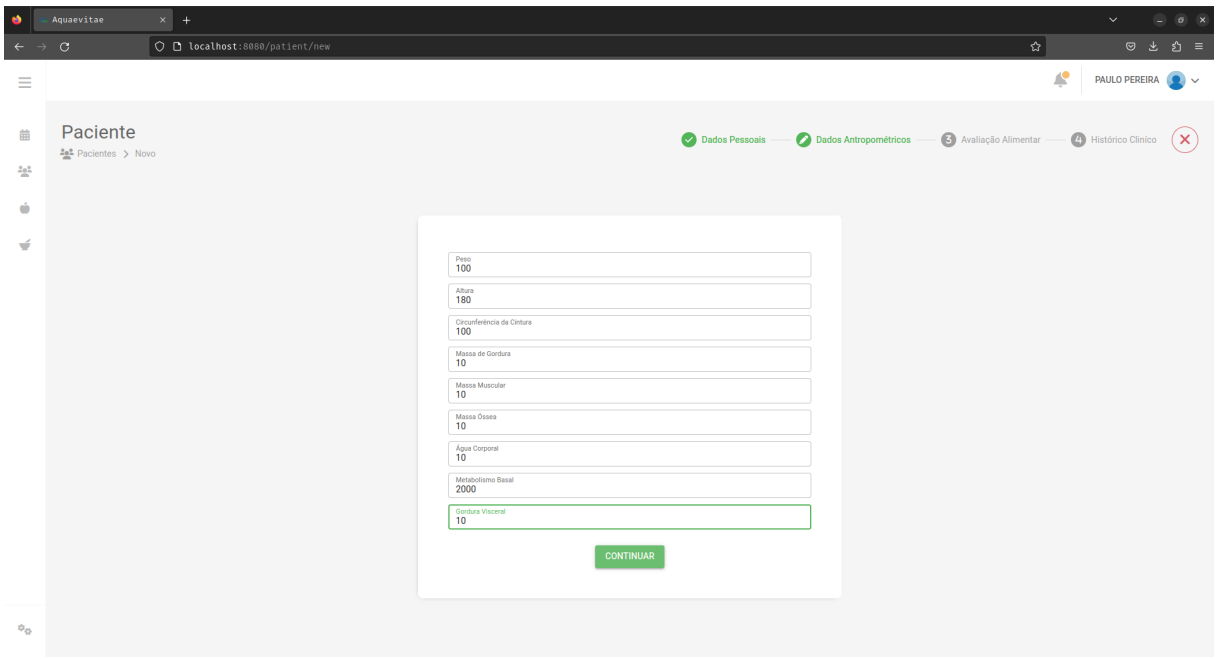



Figure 5.14: Patient File Form Example 2

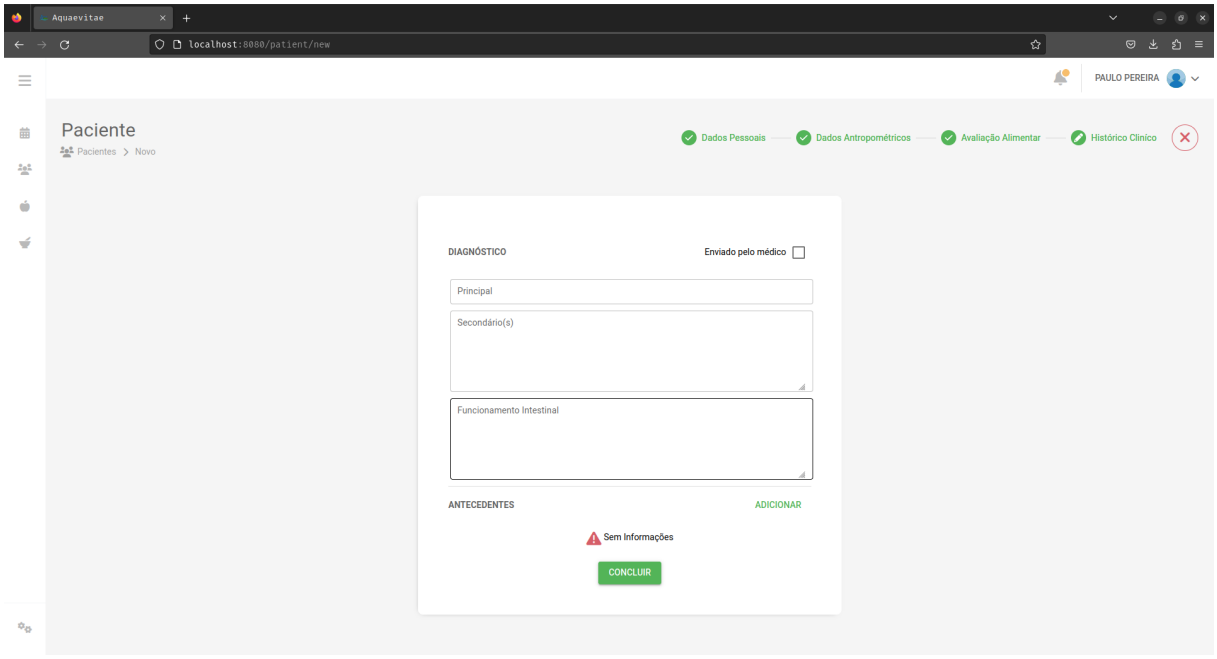



Figure 5.15: Patient File Form Example 3

All the forms (example 5.16) within our application are equipped with defined validation rules to ensure the integrity of the data. This means that each form checks the information entered by users to make sure it is correct and fits the required format. This step is crucial in maintaining accurate and reliable data throughout the application, enhancing its overall functionality and reliability.

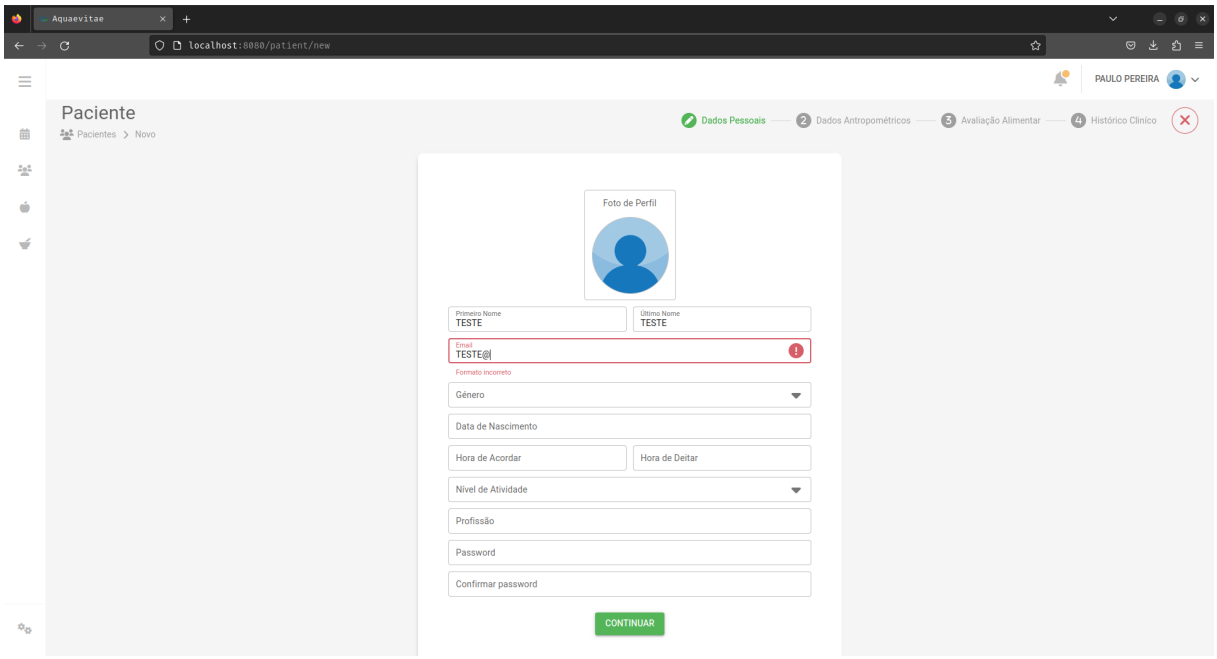



Figure 5.16: Form Validation

## 5.2 Conclusions

In conclusion, based on these results, we believe that we have successfully met both the initial requirements and the initial mockups envisioned for the web version. While we acknowledge that the final outcomes are not identical to the initial mockups, this divergence arises from our efforts to address some early shortcomings and to simplify the application's design. This was done with the aim of enhancing its intuitiveness and user-friendliness. Our adjustments reflect a thoughtful balance between adhering to our initial plan and making necessary improvements for a more efficient and user-centric experience.

# Chapter 6

# Conclusions

In this project, we aimed to create a system that works on both the web and mobile platforms. In the scope of this project, we have mainly focused on the web application. The application is designed to fulfil the requirements of the AquaeVitae project's nutrition team. Overall, We were able to deliver an application with an intuitive layout, an appealing design, and which facilitates the work of the nutrition team.

Regarding the coding part, we've built a system that can grow and change easily. This means that adding new parts to our web application is simple, and we can use a lot of the components we've already made. In the long run, this represents an advantage that saves time and effort.

In conclusion, our project has successfully achieved its fundamental objective of creating an almost ready-to-use application that allows the nutrition team to include novel thermal-based food, which is the main goal of the AquaeVitae project.

## 6.1 Future Works

Below we summarise relevant aspects that may be taken into consideration in the future versions of the web and mobile apps.

- **Testing Both Apps Together:** We think it's a good idea to test the web and

mobile apps together extensively. This approach helps in identifying and resolving any inconsistencies or issues. Continuous monitoring of their interoperability is also crucial for promptly addressing any emerging challenges.

- **Better Reports for Nutritionists:** Recognizing the need for nutritionists to have access to clear and actionable information, we propose enhancing the reporting features. This could include the integration of charts that track patient progress over time, such as variations in weight or dietary patterns. Additionally, enabling nutritionists to customize the data they view in these reports would significantly increase their utility.

- **Using WebSocket for Quick Notifications:** Implementing WebSocket for the rapid transmission of notifications could greatly improve communication. This technology would allow instant updates to be sent to users on the mobile app regarding appointment bookings, alterations, or cancellations.

We believe these simple, yet effective modifications, will greatly enhance the functionality and user experience of our web and mobile applications, benefiting both nutritionists and their patients.

# Bibliography


[1] *Aquae vitae – água termal como fonte de vida e saúde*. [Online]. Available: https://aquavalor.pt/2022/03/29/aquae-vitae-agua-termal-como-fonte-de-vida-e-saude/.

[2] *Alimentos funcionais e nutracêuticos de base termal*. [Online]. Available: https://aquaevitae.pt/alimentos-funcionais-e-nutraceuticos-de-base-termal/.

[3] F. Ferreira, J. M. Ueda, T. F. Silveira, *et al.*, "Traditional breads prepared with portuguese thermal mineral waters: Effects on centesimal and mineral composition," 2022. [Online]. Available: http://hdl.handle.net/10198/27112.

[4] H. S. Marcuzzo, *Implementation of a thermal-based food recommendation system*, 2023. [Online]. Available: http://hdl.handle.net/10198/28652.

[5] Microsoft, *Typescript*. [Online]. Available: https://www.typescriptlang.org/docs/.

[6] *Typescript github repository*. [Online]. Available: https://github.com/microsoft/TypeScript.

[7] B. A. Syed, *TypeScript Deep Dive*. GitBook, 2020. [Online]. Available: https://basarat.gitbook.io/typescript/.

[8] Y. Fain and A. Moiseev, *TypeScript Quickly*. Manning Publications, 2020.

[9] *Node.js foundation*. [Online]. Available: https://nodejs.org/en/.

[10] T. Hughes-Croucher and M. Wilson, *Node: Up and Running*. O'Reilly Media, 2018.

[11] M. Cantelon, M. Harter, T. Holowaychuk, and N. Rajlich, *Node.js in Action*. Manning Publications, 2020.

[12] D. Flanagan, *JavaScript: The Definitive Guide*. O'Reilly Media, 2020.

[13] *Node.js github repository*. [Online]. Available: `https://github.com/nodejs/node`.

[14] *Vue.js documentation*. [Online]. Available: `https://vuejs.org/v2/guide/`.

[15] E. You, *Vue.js: A (Re)introduction*. Vue.js BlogeScript, 2015. [Online]. Available: `https://blog.evanyou.me/`.

[16] C. Macrae, *Vue.js: Up and Running*. O'Reilly Media, 2020.

[17] A. Imsirovic, *Vue.js: Quick Start Guide*. Packt Publishing, 2018. [Online]. Available: `https://github.com/packtpublishing/vue.js-quick-start-guide`.

[18] *Vue.js github repository*. [Online]. Available: `https://github.com/vuejs/vue`.

[19] *Axios documentation*. [Online]. Available: `https://axios-http.com/docs/intro`.

[20] *Axios github repository*. [Online]. Available: `https://github.com/axios/axios`.

[21] R. WIERUCH, *The Road to React*. Independently published, 2020.

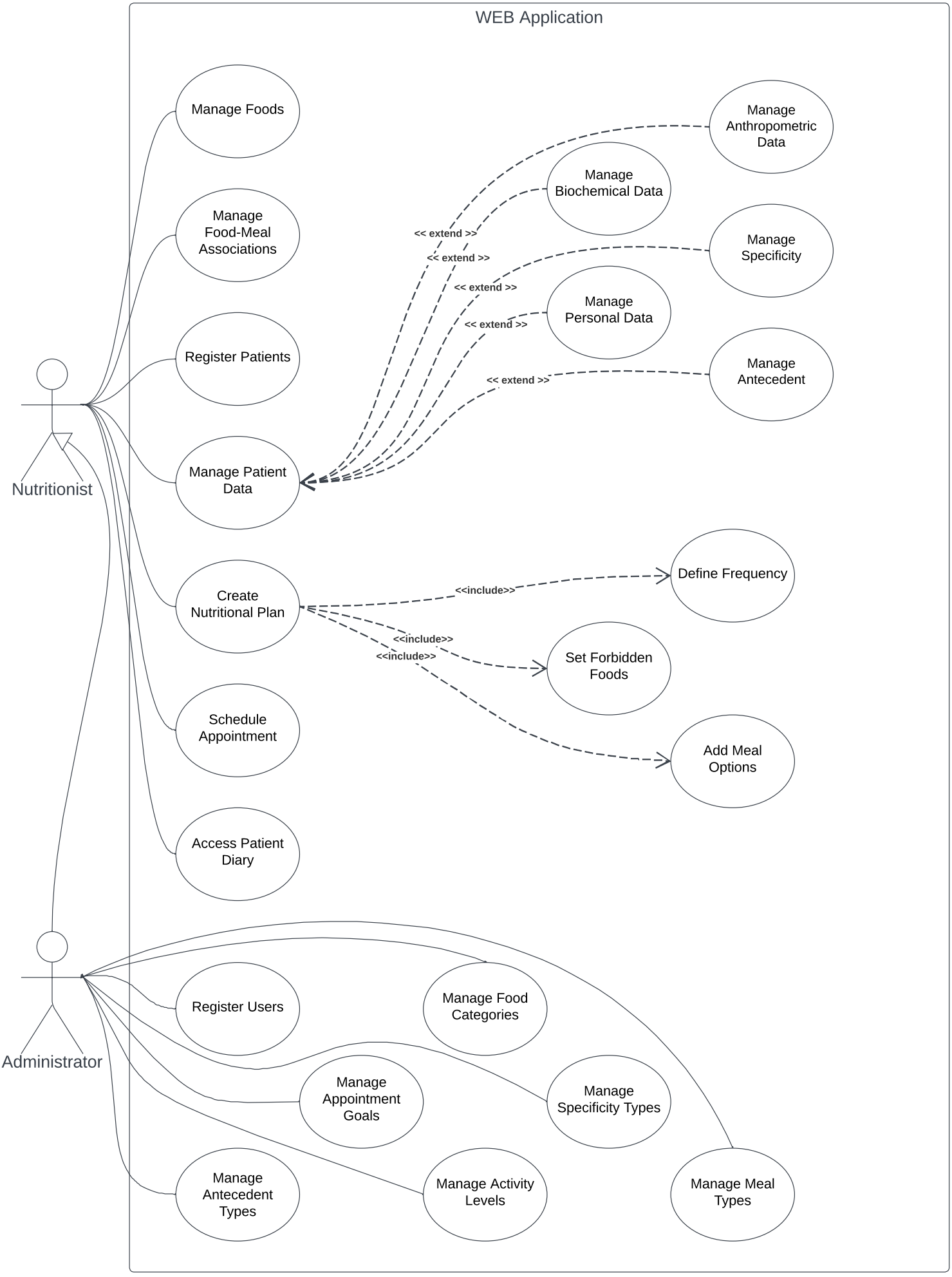
[22] *Quasar documentation*. [Online]. Available: `https://quasar.dev`.

[23] *Quasar github repository*. [Online]. Available: `https://github.com/quasarframework/quasar`.

[24] *Aquaevitae web github repository*. [Online]. Available: `https://github.com/Prica25/aquavitae-web`.

# Appendix A

# Use Cases Diagram


WEB Application
Manage Foods
Manage Food-Meal Associations
Register Patients
Manage Patient Data
Create Nutritional Plan
Schedule Appointment
Access Patient Diary
Manage Anthropometric Data
Manage Biochemical Data
Manage Specificity
Manage Personal Data
Manage Antecedent
<< extend >>
<< extend >>
<< extend >>
<< extend >>
<< extend >>
Define Frequency
Set Forbidden Foods
Add Meal Options
<<include>>
<<include>>
<<include>>
Nutritionist
Administrator
Register Users
Manage Appointment Goals
Manage Antecedent Types
Manage Food Categories
Manage Specificity Types
Manage Activity Levels
Manage Meal Types

# Appendix B

# Base Component

```
<template>
  <div v-if="!noHeader" class="content-header row items-center">
    <div style="display: flex; flex-direction: column">
      <h4 class="title">{{ title }}</h4>
      <q-breadcrumbs
        v-if="breadcrumbs.length > 0"
        class="text-grey"
        style="margin-top: 4px"
      >
        <template v-slot:separator>
          <q-icon size="1em" name="fa-solid fa-chevron-right" />
        </template>

        <q-breadcrumbs-el
          v-for="bc in breadcrumbs"
          :icon="(bc as any).icon ? `fa-solid fa-${(bc as any).icon}` : undefined"
          :label="(bc as any).label"
          :to="{ name: (bc as any).href, params: (bc as any).params }"
        />
      </q-breadcrumbs>
    </div>

    <div class="q-space"></div>
    <slot name="right-header"></slot>
  </div>
  <div v-if="noHeader && breadcrumbs.length > 0" class="content-header row">
    <q-breadcrumbs v-if="breadcrumbs.length > 0" class="text-grey">
      <template v-slot:separator>
        <q-icon size="1em" name="fa-solid fa-chevron-right" />
      </template>

      <q-breadcrumbs-el
        v-for="bc in breadcrumbs"
        :icon="(bc as any).icon ? `fa-solid fa-${(bc as any).icon}` : undefined"
        :label="(bc as any).label"
        :to="{ name: (bc as any).href, params: (bc as any).params }"
      />
    </q-breadcrumbs>
```

```
39    </div>
40    <div
41      :class="contentClass"
42      :style="`flex-direction: ${direction}; display: ${display}`"
43    >
44      <slot name="content"></slot>
45    </div>
46  </template>
47  <script lang="ts">
48  import { defineComponent } from 'vue'
49  export default defineComponent({
50    props: {
51      title: {
52        type: String,
53        required: true,
54      },
55      verticalAlignment: {
56        type: String,
57        default: 'start',
58      },
59      horizontalAlignment: {
60        type: String,
61        default: 'start',
62      },
63      noHeader: {
64        type: Boolean,
65        default: false,
66      },
67      direction: {
68        type: String,
69        default: 'row',
70      },
71      display: {
72        type: String,
73        default: 'flex',
74      },
75      breadcrumbs: {
76        type: Array,
77        default: () => [],
78      },
79    },
80    computed: {
81      contentClass() {
82        return `content-slot justify-${this.horizontalAlignment} items-${this.verticalAlignment} ${this.
              direction}`
83      },
84    },
85  })
86  </script>
```

# Appendix C

# Table Component

```
<template>
  <div class="table-wrapper">
    <table>
      <tr
        :class="`row table-header ${tableHeaderClass ? tableHeaderClass : ''}`"
      >
        <th
          v-for="column in columns"
          class="col"
          :style="`text-align: ${column.align || 'left'}; ${
            column.size ? 'flex: ' + column.size : ''
          } ${column.headerStyle ? column.headerStyle : ''}`"
          @click="sortBy(column)"
        >
          {{ column.label }}
          <q-icon
            v-if="'sortable' in column ? column.sortable : true"
            v-show="sort.by === column.name"
            :color="sort.by === column.name ? 'primary' : 'grey-6'"
            :name="`fa-solid fa-arrow-${
              sort.by === column.name ? ['up', 'down'][+sort.descending] : 'up'
            }`"
          />
        </th>
      </tr>
      <tr
        v-if="loading"
        v-for="index in pagination.rowsPerPage"
        class="row box-default"
        style="margin: 12px 0"
      >
        <td
          class="col"
          colspan="100"
          style="padding: 0; border-radius: inherit"
        >
          <q-skeleton
            animation="wave"
```

```
            height="45px"
            style="border-radius: inherit"
          />
        </td>
      </tr>
      <tr v-if="!loading" v-for="row in rows" class="row table-row box-default">
        <td
          v-for="column in columns"
          class="col"
          :style="`text-align: ${column.align || 'left'}; ${
            column.size ? 'flex: ' + column.size : ''
          } ${column.style ? column.style : ''}`"
          :title="
            disableTooltip
              ? undefined
              : typeof column.field === 'function'
              ? column.field(row)
              : row[column.name]
          "
        >
          <slot :name="column.name" :row="row">
            <span
              v-html="
                typeof column.field === 'function'
                  ? column.field(row)
                  : row[column.name]
              "
            >
            </span>
          </slot>
        </td>
      </tr>
      <tr
        v-if="!loading && rows.length === 0"
        class="row table-row box-default"
      >
        <td class="col" colspan="100" style="text-align: center">
          <slot name="no-data">
            <q-icon
              color="negative"
              name="fa-solid fa-triangle-exclamation"
              size="24px"
              style="margin-right: 4px"
            />
            {{ noDataLabel }}
          </slot>
        </td>
      </tr>
      <div class="row table-footer">
        <q-pagination
          v-model="pagination.page"
          @update:model-value="$emit('request')"
          color="primary"
          :max="pagination.pagesNumber"
          :max-pages="7"
          boundary-numbers
        />
      </div>
    </table>
```

```
  </div>
</template>
<script lang="ts">
import { defineComponent } from 'vue'

export default defineComponent({
  emits: ['update:pagination', 'update:sort', 'request'],
  props: {
    pagination: {
      type: Object,
      default: () => ({
        page: 1,
        rowsPerPage: 8,
        pagesNumber: 0,
        rowsNumber: 0,
      })
    },
    sort: {
      type: Object,
      default: () => ({
        by: null as null | String,
        descending: false,
      })
    },
    tableHeaderClass: String,
    noDataLabel: {
      type: String,
      default: 'Sem resultados'
    },
    loading: {
      type: Boolean,
      default: false,
    },
    columns: {
      type: Array<any>,
      default: () => [],
    },
    rows: {
      type: Array<any>,
      default: () => [],
    },
    disableTooltip: {
      type: Boolean,
      default: false,
    }
  },
  data() {
    return {
      rowsSorted: null as null | Array<any>,
    }
  },
  methods: {
    sortBy(column: any) {
      if ('sortable' in column ? column.sortable : true) {
        if (this.sort.by !== column.name) {
        this.sort.by = column.name
        this.sort.descending = false
      } else {
        this.sort.descending = !this.sort.descending
```

```
157         }
158
159         this.$emit('request')
160         }
161
162       },
163     },
164 })
165 </script>
```

# Appendix D

# Autocomplete Component

```
<template>
  <q-select
    dense
    outlined
    v-model="value"
    :options="options"
    :loading="isLoading"
    @virtual-scroll="onScroll"
    :label="label"
    :option-value="valueKey"
    :option-label="labelKey"
    emit-value
    map-options
    use-input
    @filter="onFilter"
  />
</template>
<script lang="ts">
import { defineComponent } from 'vue'
import type ResponseList from '@/types/ResponseList'

export default defineComponent({
  props: {
    modelValue: {
      type: String,
      required: true,
    },
    type: {
      type: String,
      required: true,
    },
    valueKey: {
      type: String,
    },
    labelKey: {
      type: String,
      default: false,
    },
```

```
    label: {
      type: String,
      default: 'Procurar...',
    },
    defaultFilter: {
      type: String,
    },
  },
  emits: ['update:modelValue'],
  data() {
    return {
      SERVICE: import(
        /* @vite-ignore */ `../../services/${this.type}Service.ts`
      ) as any,
      options: [] as any,
      pagination: {
        sortBy: this.labelKey,
        descending: false,
        page: 1,
        rowsPerPage: 20,
        pagesNumber: 0,
        rowsNumber: 0,
      },
      isLoading: true,
      filter: null as string | null,
    }
  },
  async mounted() {
    this.SERVICE = (await this.SERVICE).default
    let response = (
      await this.SERVICE.index(
        this.pagination.page,
        1,
        `${this.pagination.sortBy}:${
          this.pagination.descending ? 'DESC' : 'ASC'
        }`,
        this.labelKey,
        this.filters
      )
    ).data as ResponseList

    this.pagination.rowsNumber = response.count
    this.pagination.pagesNumber = response.last_page
    this.options = Object.freeze(response.data)
    this.isLoading = false
  },
  computed: {
    value: {
      get(): string {
        return this.modelValue
      },
      set(val: string): void {
        this.$emit('update:modelValue', val)
      },
    },
    filters() {
      const fil = [
        this.defaultFilter,
        this.filter ? `${this.labelKey}:${this.filter}` : null,
```

```
    ].filter((value) => !!value)

    if (fil.length > 0) {
      if (fil.length === 1) {
        return fil[0]
      }
      return fil
    }
    return null
  },
},
methods: {
  async onScroll(props: any) {
    if (
      !this.isLoading &&
      this.pagination.page < this.pagination.pagesNumber &&
      props.index >= this.options.length - 10
    ) {
      this.isLoading = true
      this.pagination.page++

      let response = (
        await this.SERVICE.index(
          this.pagination.page,
          this.pagination.rowsPerPage,
          `${this.pagination.sortBy}:${
            this.pagination.descending ? 'DESC' : 'ASC'
          }`,
          this.labelKey,
          this.filters
        )
      ).data as ResponseList

      this.options = Object.freeze([...this.options, ...response.data])
      this.isLoading = false
    }
  },
  async onFilter(str: string, update: (fnc: () => void) => void) {
    this.isLoading = true
    this.filter = str
    let response = (
      await this.SERVICE.index(
        this.pagination.page,
        this.pagination.rowsPerPage,
        `${this.pagination.sortBy}:${
          this.pagination.descending ? 'DESC' : 'ASC'
        }`,
        this.labelKey,
        this.filters
      )
    ).data as ResponseList

    this.pagination.rowsNumber = response.count
    this.pagination.pagesNumber = response.last_page
    this.pagination.page = 1

    update(() => {
      this.options = Object.freeze(response.data)
    })
```

```
157         this.isLoading = false
158       },
159     },
160 })
161 </script>
```